Kapitel 7 Appendix

7.1 Quellencode des Merkmal Extraktion Software.

7.1.1 Image.h

```
/***************************************************************************
                          image.h  -  description
                             -------------------
    begin                : Sun Apr 13 2003
    copyright            : (C) 2003 by root
    email                : root@localhost.localdomain
 ***************************************************************************/

/***************************************************************************
*                                                                          *
* This program is free software; you can redistribute it and/or modify  *
* it under the terms of the GNU General Public License as published by  *
* the Free Software Foundation; either version 2 of the License, or     *
* (at your option) any later version.                                   *
*                                                                          *
***************************************************************************/

 //Header file for image recognition thing
#include <math.h>

#define WINDOWS_Y             5 // I have set it at 5 because it said so in the paper
#define FALSE                       0
#define TRUE                        1
struct dat_point
{
int x;
int y;
} ;


struct datanow
{
double north;                  //will contain the number of counted segments.
double south;
double east;
double west;
} ;

struct stats
{
double min;
double max;
double std;
double avg;
double w_min;
double w_max;
};

 struct window
 {
    int pos_x;
    int pos_y;
    int size_x;
    int size_y;
    struct datanow dat;
```

```
 };

struct global_stats_and_errors
{
double max_n;
double max_e;
double max_s;
double max_w;
long n_hits[1001];
long e_hits[1001];
long s_hits[1001];
long w_hits[1001];
int enabled;

};
 struct windows
 {
 int  pos_x;
 int  pos_y;
 int  windows_x;
 int  windows_y;
 struct window **win;
 struct stats stat_n;           //statistics on window
 struct stats stat_e;           //statistics on window
 struct stats stat_s;           //statistics on window
 struct stats stat_w;           //statistics on window

 int error_in_output;         //stores the error in the output window
 };


void set_window_size_y(struct windows* win,int new_size);
void set_window_size_x(struct windows* win,int new_size);
void set_window_pos_x(struct windows* win,float offset);
void set_window_pos_y(struct windows* win,float offset);
int inside_box(struct dat_point *one,struct dat_point *two,int x ,int y) ;
int inside_window(struct window *win,int x,int y);
double modu(double in);
void swap_and_set_primary_point(struct dat_point* begin,struct dat_point* one,struct
dat_point* two);
unsigned char double_to_char(double input,struct windows* win);
//out of saveaspic.c
void save_data_tiff(char* file,struct windows* win,int sq_width,int sq_height,char*
label);
void fill_rect(int in_x,int in_y,int size_x,int size_y,unsigned char **array,unsigned
char val);
void save_gnu_plot_skpt_file(char* filename_data,char* filename,struct windows* win,int
output_window_height);
void make_script_gnuplot_to_pmg(char* filename,char* dat);
void extract_file_name_from_path(char* in,char* out);


//out of functions.c
void reset_window_data_struct(struct windows* win);
void get_distance_between_two_points_and_store_in_window(struct window *win,struct
dat_point* one,struct dat_point* two);
void surch_windows_for_line_inside(struct windows* win,struct dat_point *one,struct
dat_point *two);

//out of norm.c
void norm_data_to_window_hight(struct windows* win);
void norm_data_to_redy_for_image_transfer(struct windows* win);
void get_avrage_of_directions(struct windows* win);
void get_std_of_data(struct windows* win);
void get_min_value(struct windows* win);
void get_max_value(struct windows* win,struct global_stats_and_errors *errors);
void init_stats(struct windows* win,double V,struct global_stats_and_errors *errors);
void w_min(struct windows* win,double V);
void w_max(struct windows* win,double V);
double x_norm(struct windows* win,double in);
double normme(struct windows* win,double in,double max);
void norm_data_set(struct windows* win);
```

```
//help functions to go wiht main
void printlots(char **msg);
void printintro();
void printhelp();
void dotdot();

//sklintf
void anal_file(char *inputfile,char *outputfile,char onlytif,struct
global_stats_and_errors *errors);
```

7.1.2 help.c

```
/***************************************************************************
                          help.c  -  description
                             -------------------
    begin                : Tue Apr 22 2003
    copyright            : (C) 2003 by
    email                :
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
#include <stdio.h>
#include "image.h"
char *msggnu[] =
{ "*************************************************************************",
                  "*
*",
                  "*   This program is free software; you can redistribute it and/or
modify  *",
                  "*   it under the terms of the GNU General Public License as published
by  *",
                  "*   the Free Software Foundation; either version 2 of the License, or
*",
                  "*   (at your option) any later version.
*",
                  "*
*",
                  "*************************************************************************
******",{0}};


char *msghelp[] = {"usage image (-@) inputfile",
                  "output files set as defult as inputfile.tif, inputfile.skpt
,inputfile.png",
                  ".tif is a tif format image , skpt is a gnuplot script and png is a
gnuplot compatible data table",{0}};

 void printlots(char **msg)
 {
   int i=0;
   do
   {
       printf("%s\n",msg[i]);
   }
   while(msghelp[++i]);
 }

 void printintro()
 {
```

```
  printlots(msggnu);
 }

 void printhelp()
 {
   printlots(msghelp);
 }

 void dotdot()
 {
   putchar('.');
   putchar('.');
 }
```

7.1.3 functions.c

```
/***************************************************************************
                          functions.c  -  description
                             -------------------
    begin                : Sun Apr 13 2003
    copyright            : (C) 2003 by Roderick MacKenzie
    email                : eeyurcim@nottingham.ac.uk
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
#include <stdio.h>
#include <math.h>
#include "image.h"



void set_window_size_y(struct windows* win,int new_size)
{
  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    win->win[x][y].size_y=new_size;
    }
  }

}

void set_window_size_x(struct windows* win,int new_size)
{
  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    win->win[x][y].size_x=new_size;
    }
  }

}


void set_window_pos_x(struct windows* win,float offset)
```

```
{
  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    win->win[x][y].pos_x=x*win->win[x][y].size_x*offset;
    }
  }

}

void set_window_pos_y(struct windows* win,float offset)
{
  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    win->win[x][y].pos_y=y*win->win[x][y].size_y*offset;
    }
  }

}

void reset_window_data_struct(struct windows* win)
{
  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    win->win[x][y].dat.north=0;
    win->win[x][y].dat.east=0;
    win->win[x][y].dat.south=0;
    win->win[x][y].dat.west=0;

    }
  }

}


void debug_print_windows(struct windows* win)
{
  int x=0;
  int y=0;
    for (y=0;y<win->windows_y;y++)
    {
    for (x=0;x<win->windows_x;x++)
    {
    printf("%i %i %i n%.0f e%.0f s%.0f w%.0f\n",x,y,win->win[x][y].pos_x,win->win[x]
[y].pos_y,win->win[x][y].dat.north,win->win[x][y].dat.east,win->win[x][y].dat.south,win-
>win[x][y].dat.west);
    }
    printf("nlnlnlnlnlnlnlnlnlnlnl\n");
    }
}

void surch_windows_for_line_inside(struct windows* win,struct dat_point *one,struct
dat_point *two)
{
  int x=0;
  int y=0;
  int inter_y=0;      //the intersection of y
  int inter_x=0;      //the intersection of y
```

```
  struct dat_point pa[5];     //point array
  int pac=0;        //point array
    for (y=0;y<win->windows_y;y++)
      for (x=0;x<win->windows_x;x++)
      {
         pac=0;       //reset the data array
        //both in side
        if( ((one->y>win->win[x][y].pos_y)&&(one->y<win->win[x][y].pos_y+win->win[x]
[y].size_y))&&
        ((one->x>=win->win[x][y].pos_x)&&(one->x<=win->win[x][y].pos_x+win->win[x]
[y].size_x))&&
        ((two->x>=win->win[x][y].pos_x)&&(two->x<=win->win[x][y].pos_x+win->win[x]
[y].size_x))&&
        ((two->y>win->win[x][y].pos_y)&&(two->y<win->win[x][y].pos_y+win->win[x]
[y].size_y))        )
              {   //just store the points
                   pa[pac].x=one->x;
                   pa[pac++].y=one->y;
                   pa[pac].x=two->x;
                   pa[pac++].y=two->y;
              }  else   //not both in
              {
                //both outside
               // if( ((one->y<win->win[x][y].pos_y)&&(one->y>win->win[x][y].pos_y+win-
>win[x][y].size_y))&& //check if y is out side
              //     ((one->x<=win->win[x][y].pos_x)&&(one->x>=win->win[x][y].pos_x+win-
>win[x][y].size_x))) //check if x is out side



                   //dose it cross LEFT
                if (two->x-one->x)    //catch devide by 0 errors
                {
                   inter_y= one->y+(win->win[x][y].pos_x-one->x)*(two->y-one->y)/(two-
>x-one->x);
                   if (inside_box(one,two,win->win[x][y].pos_x,inter_y))
                   if ((inter_y>win->win[x][y].pos_y)&&(inter_y<win->win[x]
[y].pos_y+win->win[x][y].size_y))
                   {
                       pa[pac].x=win->win[x][y].pos_x;
                       pa[pac++].y=inter_y;
                   }

                   //right hand side of the box
                   inter_y= one->y+(win->win[x][y].pos_x-one->x+win->win[x]
[y].size_x)*(two->y-one->y)/(two->x-one->x);
                   if (inside_box(one,two,win->win[x][y].pos_x+win->win[x]
[y].size_x,inter_y))
                   if ((inter_y>win->win[x][y].pos_y)&&(inter_y<win->win[x]
[y].pos_y+win->win[x][y].size_y))
                   {
                   pa[pac].x=win->win[x][y].pos_x+win->win[x][y].size_x;
                   pa[pac++].y=inter_y;
                   }
               }

                if (two->y-one->y)    //catch devide by 0 errors
                {
                   //top
                   inter_x= one->x+(win->win[x][y].pos_y-one->y)*(two->x-one->x)/(two-
>y-one->y);
                   if (inside_box(one,two,inter_x,win->win[x][y].pos_y))
                   if ((inter_x>=win->win[x][y].pos_x)&&(inter_x<=win->win[x]
[y].pos_x+win->win[x][y].size_x))
                   {
                     pa[pac].x=inter_x;
                      pa[pac++].y=win->win[x][y].pos_y;
                   }

                   ///bottom
                   inter_x= one->x+(win->win[x][y].pos_y-one->y+win->win[x]
[y].size_y)*(two->x-one->x)/(two->y-one->y);
```

```
                    if (inside_box(one,two,inter_x,win->win[x][y].pos_y+win->win[x]
[y].size_y))
                    if ((inter_x>=win->win[x][y].pos_x)&&(inter_x<=win->win[x]
[y].pos_x+win->win[x][y].size_x))
                    {
                      pa[pac].x=inter_x;
                       pa[pac++].y=win->win[x][y].pos_y+win->win[x][y].size_y;
                    }
                 }

              }   //end of else statement

                 // check to see how many points we have found
                 if (pac==1)      //if we have only only found one intersection of our
box that means one point is inside of our box
                 {
                    //so we start off with the precondition that only one point can be in
the window
                    if (inside_window(&win->win[x][y],one->x,one->y))
                    {     //one is in side the window
                       pa[pac].x=one->x;
                       pa[pac++].y=one->y;
                    }

                    if (inside_window(&win->win[x][y],two->x,two->y))
                    {     //two is in side the window
                       pa[pac].x=two->x;
                       pa[pac++].y=two->y;
                   }


              }           //end of the pac check

              if (pac==0)
              {
              //noting found
              }

              if (pac==2)         //ie we have just found 2  exactley as we want
              {
                //get length
                //rember there is an error here
                swap_and_set_primary_point(one,&pa[0],&pa[1]);
                get_distance_between_two_points_and_store_in_window(&win->win[x]
[y],&pa[0],&pa[1]) ;
              }


      }         //end of the for loops

}
/*

       //it is silley to continue if both x values are smaller or both bigger
       //if ((one->x<win->win[x][y].pos_x)&&(two->x<win->win[x][y].pos_x) ||(one->x>win-
>win[x][y].pos_x)&&(two->x>win->win[x][y].pos_x))
       //{

       //}
       while(1)
       {
       //if  (one->x<win->win[x][y].pos_x)&&(two->x<win->win[x][y].pos_x)  break;
//if both points are smaller exit (x)
       //if   (one->x>win->win[x][y].pos_x)&&(two->x>win->win[x][y].pos_x)  break;
//if both points are biger exit    (y)
       //if  (one->y<win->win[x][y].pos_y)&&(two->y<win->win[x][y].pos_y)  break;
//if both points are smaller exit (y)
       //if   (one->y>win->win[x][y].pos_y)&&(two->y>win->win[x][y].pos_y)  break;
//if both points are biger exit    (y)


        if (two->x-one->x)    //catch devide by 0 errors
```

```
          {
                              //both the points are in side our square
                              if (((one->y>win->win[x][y].pos_y)&&(one->y<win->win[x]
[y].pos_y+win->win[x][y].size_y))
                             &&((two->y>win->win[x][y].pos_y)&&(two->y<win->win[x]
[y].pos_y+win->win[x][y].size_y)))
                              {
                              crossses=TRUE;
                              length = sqrt(((one->x-two->x)*(one->x-two->x) +(one->y-two-
>y)*(one->x-two->x));
                              angle = (two->y-one->y)/(two->x-one->x);
                              break;
                              }


                         }


       break;     //should not be exercuted
       }




          //left hand side of the box
          inter_y= (one->x+win->win[x][y].pos_x)*((two->y-one->y)/(two->x-one->x))-one->y;
                    if (intey_y>win->win[x][y].pos_y)&&(intey_y<win->win[x][y].pos_y+win-
>win[x][y].size_y)
                    {
                      crosses=TRUE;
                      }

          //right hand side of the box
          inter_y= (one->x+win->win[x][y].pos_x+win->win[x][y].size_x)*((two->y-one->y)/
(two->x-one->x))-one->y;
                    if (intey_y>win->win[x][y].pos_y)&&(intey_y<win->win[x][y].pos_y+win-
>win[x][y].size_y)
                    {
                      crosses=TRUE;
                      }


          }//       end of if (two->x-one->x)
          else    //a ha we have corhgt a devide by 0 arnt we luckey
          {
                         if ((one->x>win->win[x][y].pos_x)&&(one->x<win->win[x]
[y].pos_x+win->win[x][y].size_x))       //is the vertical line in our box
                         {
                                corsses=TRUE;         //it is in our box
                         }

          };

*/
int inside_box(struct dat_point *one,struct dat_point *two,int x ,int y)      //is a
point in side a box defined by 2 points
{
  //test y first
int ok=FALSE;
if (one->y>two->y)
{
  if (y<=one->y)
  if (y>=two->y)
      ok=TRUE;
}else
{
  if (y>=one->y)
  if (y<=two->y)
      ok=TRUE;
```

```
}

if (ok==TRUE)
{
  ok=FALSE;
  if (one->x<two->x)
  {
   if (x>=one->x)
   if (x<=two->x)
      ok=TRUE;
  }else
  {
   if (x<=one->x)
   if (x>=two->x)
       ok=TRUE;
   }
}

return ok;
}

int inside_window(struct window *win,int x,int y)
{

if (x>=win->pos_x)
if (x<=win->pos_x+win->size_x)
if (y>win->pos_y)
if (y<win->pos_y+win->size_y)
return TRUE;

return FALSE;
}


void get_distance_between_two_points_and_store_in_window(struct window *win,struct
dat_point* one,struct dat_point* two)
{
double dist_x=two->x-one->x;
double dist_y=one->y-two->y;
//double dist = sqrt(dist_x*dist_x+dist_y*dist_y);
//double ang=0;
//if (dist_x!=0)
//{
//ang=atan(sqrt((dist_y*dist_y)/(dist_x*dist_x)));
//}
//work out which quadrant we are in
//figure out top bottom first
  if (dist_y>=0)   //top
  {
      if (dist_x>0)   //right
      {
          win->dat.north+=modu(dist_y);
          win->dat.east+=modu(dist_x);
      }

      if (dist_x<0)          //left
      {
          win->dat.north+=modu(dist_y);
          win->dat.west+=modu(dist_x);
      }

      if (dist_x==0)   //it is 0 caught devide by 0
      {
         win->dat.north+=modu(dist_y);
      }

  }else   //bottom
  {
      if (dist_x>0)   //right
      {
          win->dat.south+=modu(dist_y);
```

```
          win->dat.east+=modu(dist_x);
      }

      if (dist_x<0)          //left
      {
          win->dat.south+=modu(dist_y);
          win->dat.west+=modu(dist_x);
      }

      if (dist_x==0)   //it is 0 caught devide by 0
      {
         win->dat.south+=modu(dist_y);
      }


  }


}

double modu(double in)
{
if (in<0)
{return in*=-1;}else
{
return in;}
}

void swap_and_set_primary_point(struct dat_point* begin,struct dat_point* one,struct
dat_point* two)
{
double one_dist = sqrt((one->x-begin->x)*(one->x-begin->x) + (one->y-begin->y)*(one->x-
begin->y));                    //rember we are missing the square root
double two_dist = sqrt((two->x-begin->x)*(two->x-begin->x) + (two->y-begin->y)*(two->x-
begin->y));                   //rember we are missing the square root
int temp_x=0;
int temp_y=0;
  if (two_dist<one_dist)      //if they are the wrong way around swap them
  {
      temp_x=one->x;
      temp_y=one->y;
      one->x=two->x;
      one->y=two->y;
      two->x=temp_x;
      two->y=temp_y;
  }

}
```

7.1.4 sklintf.c

```
/***************************************************************************
                          sklintf.c  -  description
                             -------------------
    begin                : Tue Apr 22 2003
    copyright            : (C) 2003 by
    email                :
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "usralloc.h"
```

```
#include "argio.h"
#include "ascio.h"
#include "fname.h"
#include "cmdlnerr.h"
#include "skelio.h"
#include "image.h"

static int  windows_y = WINDOWS_Y;

void anal_file(char *inputfile,char *outputfile,char onlytif,struct
global_stats_and_errors *errors)
{
  short i,k,kk;
  char filename_data[255];                                    // a temmpory filename
buffer.
  char filename_skpt[255];                                    // a temmpory filename
buffer.
  char filename_image[255];                                    // a temmpory filename
buffer.

    TSKELETT skel;
    SKL_FILE *inf=0;                                             //skelet file
    FILE *outf=0;                                                //the gnuplot masta
data file
// char temp2[255];
//    extract_file_name_from_path(dat,temp2);

    struct  windows thiswindow;                                   //define the data struct

    int y=0;
    int x=0;
    int static stroke=20;                                         //hard set for the time
beaing
    float static overlap=0.5;                                     //set of the over lap


    strcpy(filename_data,outputfile);                            //set up file names
    strcat(filename_data,".plt");                                //set the output file name
... plot file

    strcpy(filename_skpt,outputfile);                            //set up file names
    strcat(filename_skpt,".skpt");                               //set the output file name
... gnuscript file

    strcpy(filename_image,outputfile);                            //set up file names
    strcat(filename_image,".tif");                               //set the output file name
... tif file

    printf("\n%s",inputfile);

    if ( !(inf = SklOpen ( inputfile, SKLIO_READ, NULL, NULL )))
    {
    CmdExitMess ( CMDLN_INPUTOPEN,
                  "%s:\n%s",
                  "stdin" ,
                  SklErrorMess());
    }



    //first init the array
    thiswindow.pos_x=0;
    thiswindow.pos_y=0;
    thiswindow.windows_x=(SklPicX(inf)/stroke);          //hard set the number of windows
for the time beaing
    thiswindow.windows_y=windows_y;
    //nog grab the mem

    thiswindow.win=(struct window**) MemMalloc(sizeof(struct
window*)*thiswindow.windows_x);    //grabed the number of rows
    for (x=0;x<thiswindow.windows_x;x++)
    {
```

```
     thiswindow.win[x]=(struct window*)MemMalloc(sizeof(struct
window)*thiswindow.windows_y);
    }

    set_window_size_y(&thiswindow,SklPicY(inf)/windows_y);    //set the y size of the
window the same as the offset.... no over lap
    set_window_pos_y(&thiswindow,1);    //overlap

    set_window_size_x(&thiswindow,stroke/overlap);
    set_window_pos_x(&thiswindow,overlap);
    reset_window_data_struct(&thiswindow);
    //struct dat_point one={0,70};           //over cell  y=const
    //struct dat_point two={200,70};
    //struct dat_point one={0,70};          //under cell  y=const
    //struct dat_point two={100,70};
    //struct dat_point one={0,70};          //under cell  y=const
    //struct dat_point two={100,70};
    //struct dat_point one={0,0};          //going side to side
    //struct dat_point two={100,23};


    struct dat_point old_dat_point={-1,-1};   //on the first run through set them to -1
    struct dat_point new_dat_point={-1,-1};   //on the first run through set them to -1


    if (onlytif==FALSE)
    if ( !(outf = fopen ( filename_data, "wb" )) )       //open our output file and check
if it is OK
    {
        printf("can not open file\n");
        return 0;
    }



    if (outf)
    {
    fprintf (outf, "# Skelett als ASCII datei fuer gnuplot\n");
    fprintf (outf, "# erzeugt von image\n");
    fprintf (outf, "# Kanten plotten: plot '%s' wi li\n", filename_data );
    fprintf (outf, "# Knoten plotten: plot '%s' usi 3:4 wi points\n", filename_data );
    }

    for ( i=0 ; i< SklNoSkl(inf) ; i++ )       //loop for throught all the skelertons in
a file
    {
      if ( !SklReadNext ( inf, &skel, TRUE, TRUE ) )          //set all the pointers to
the next skl
      {
        CmdExitMess ( CMDLN_INPUTREAD,
                      "%s:\n%s",
                      "stdin",
                      SklErrorMess());
      }
                                                              //ok we are set up and
ready to go
      for ( k=0 ; k<skel.no_kanten ; k++ )                    //run throught the number
of points in the moddel
      {

        if ((k%100)==0)
        {
        dotdot();
        }

        switch ( skel.kant[k].c.type )                        //what type is our point
        {
         //eligidley there are 2 types SKL_KANT_POLY and SKL_KANT_KETTE
         case SKL_KANT_POLY:

          old_dat_point.x=skel.kant[k].c.obj.pol.point[0].x;
          old_dat_point.y=skel.kant[k].c.obj.pol.point[0].y;
```

```
          if (outf)
          {
              fprintf (outf, "%d %d ", skel.kant[k].c.obj.pol.point[0].x,
                                       skel.kant[k].c.obj.pol.point[0].y  );
              fprintf (outf, "%d %d\n", skel.knot[skel.kant[k].k[0]].x,
                                        skel.knot[skel.kant[k].k[0]].y  );
          }

          for ( kk=1 ; kk < (skel.kant[k].c.obj.pol.n - 1) ; kk++ )     //loop through
all the points
          {
              //store new points in a nice struct
              new_dat_point.x=skel.kant[k].c.obj.pol.point[kk].x;
              new_dat_point.y=skel.kant[k].c.obj.pol.point[kk].y;

              surch_windows_for_line_inside(&thiswindow,&old_dat_point,&new_dat_point);

              //store old point so that we are making the line
              old_dat_point.x=skel.kant[k].c.obj.pol.point[kk].x;
              old_dat_point.y=skel.kant[k].c.obj.pol.point[kk].y;

              if (outf)
              {

                  fprintf (outf, "%d %d 0 0\n",
                              skel.kant[k].c.obj.pol.point[kk].x,
                             skel.kant[k].c.obj.pol.point[kk].y  );
              }

          }

          if ( skel.kant[k].c.obj.pol.n > 1 )         //this seems to be a bit odd
          {

              new_dat_point.x=skel.kant[k].c.obj.pol.point[kk].x;
              new_dat_point.y=skel.kant[k].c.obj.pol.point[kk].y;

              surch_windows_for_line_inside(&thiswindow,&old_dat_point,&new_dat_point);

                  if (outf)
                  {

                    fprintf (outf, "%d %d ",
                               skel.kant[k].c.obj.pol.point[kk].x,
                              skel.kant[k].c.obj.pol.point[kk].y  );
                    fprintf (outf, "%d %d\n",
                              skel.knot[skel.kant[k].k[1]].x,
                              skel.knot[skel.kant[k].k[1]].y  );
                  }
          }

          if (outf) fprintf ( outf, "\n" );
          break;

         //I can not handle SKL_KANT_KETTE file types exit nicley
         case SKL_KANT_KETTE:
        printf("Sory this copy of this program can not handel file containing
SKL_KANT_KETTE");
        CmdExit(CMDLN_WRONGINPUTNUM);

          break;

         default:
          break;
        }
      }

      SklFree ( &skel );
    }
```

```
    if (outf)
save_gnu_plot_skpt_file(filename_data,filename_skpt,&thiswindow,SklPicY(inf));

    norm_data_to_window_hight(&thiswindow);                    //the data should now be
normilise to the hight of the window

    init_stats(&thiswindow,0.9,errors);                              //V hard set for the
time as 0.5

    norm_data_set(&thiswindow);

    save_data_tiff(filename_image,&thiswindow,1,1,SklLbl(inf));

    if (outf) make_script_gnuplot_to_pmg(outputfile,filename_skpt);




    SklClose (inf);         //close in put file
    if (outf) fclose (outf);          //close the output file

//    printf("\n\nSucess data outputed.......\n");
//    printf("Total error in output image due to truncating data
%i",thiswindow.error_in_output);


    //debug_print_windows(&thiswindow);

    //flush the memory  of our windows and big window

    for ( y=0;y<thiswindow.windows_x;y++)                       //watch out to get the
corect x or y
    {
    MemFree(thiswindow.win[y]);
    }

    MemFree(thiswindow.win);



}
```

7.1.5 main.c

```
/***************************************************************************
                          main.c  -  description
                             -------------------
    begin                : Sun Apr 13 13:22:56 BST 2003
    copyright            : (C) 2003 by root
    email                : root@localhost.localdomain
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
/*-------------------------------------------------------------------------


    parts of this program are base on SKL2PLT.C 2.10.95, rolf bippus
Roderick MacKenzie


-------------------------------------------------------------------------*/
```

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <curses.h>
#include "usralloc.h"
#include "argio.h"
#include "ascio.h"
#include "fname.h"
#include "cmdlnerr.h"
#include "skelio.h"
#include "image.h"



/*-------- global parameters --------------*/

//static char programname[] = "image";
//static char progdescription[] ="skl analizer";
//static char copyright[] = "base: IfN TU Braunschweig, Rolf Bippus, 2.10.95,updated
Roderick MacKenzie 2003";

//short arg_noprompt = FALSE;
//short arg_generate_parfile = FALSE;
//short arg_help = FALSE;
//char  *arg_parfile="";

//short arg_stdout = FALSE;
//short arg_stdin = FALSE;

//char * arg_description = NULL;
//char * arg_creation_ID = "image";

//long arg_offset[2] = { 0,0 };



int main ( int argc, char *argv[] )
{
//  char ** inputfilelist;                 //the list of input files
//  char ** outputfilelist;                //the list of output files
//  short list_len;                        //the length of the file list
//  int filecnt;

//  char ** argerr;
  char inputfile[256];
  char outputfile[256];
  char  temp[255];
  FILE *filelist=0;
  char input;
  struct global_stats_and_errors errors;
  errors.enabled=FALSE;
//  char cmdline [256];
//  char filename[128];
//  char output_file_name[128];
//  char *pc;


//  int x,y;
//  unsigned char * pcode;
  // one is input two is output
//   char path[255];
//   extract_file_name_from_path("one.txt",path);
//   strcpy(path,);
//   printf("%s",path);
//   return 0;

  if (argc==1)          //if the program is called whith out prams
  {
    printf("\n nothing to be done, no arguments\n");
    return 0;         //exit
  }
```

```
  if (!strcmp(argv[1],"-@"))
  {
    if (argc>=2)
    {
    filelist = fopen ( argv[2], "r" );       //open the list of files in read mode
    }else
    {
      printintro();        //put up the gnu info
      printhelp();
      return 0;
    }

  }else
  {
     if (argc!=3)
     {
       printintro();        //put up the gnu info
       printhelp();
       return 0;
     }
      strcpy(inputfile,argv[1]);
      strcpy(outputfile,argv[2]);

  }


  do      //the main processing loop
  {
    if (filelist)
    {
      //strcpy(inputfile,"");
      fscanf(filelist,"%s",inputfile);
      // norm index exercution -@ truncatedindex ../../results2/
      if (argc>=4)
      {
       strcpy(outputfile,argv[3]);
       extract_file_name_from_path(inputfile,temp);
       strcat(outputfile,temp);
       printf("%s.xxx",outputfile);
      }       //output file the same as the input file

      if (feof(filelist))      //if the end of the file has been reached
      {
        fclose(filelist);
        filelist=0;
        break;
      }
    }else     //if there is no file list
    {
       strcpy(outputfile,argv[2]);
       extract_file_name_from_path(inputfile,temp);
       strcat(outputfile,temp);
       printf("%s.xxx",outputfile);
    }

    anal_file(inputfile,outputfile,FALSE,&errors);
//      if( getchar()=='x')
//      {
//        printf("nicley exited\n");
//        return 0;
//      }
  }
  while(filelist!=0);
  if (errors.enabled==TRUE) printf("\nn %f \ne %f \ns %f \nw
%f\n",errors.max_n,errors.max_e,errors.max_s,errors.max_w);
  CmdExitMess(CMDLN_OK, "(%s)", "image");

  return 0;
}
```

7.1.6 saveaspic.c

```
/***************************************************************************
                          saveaspic.c  -  description
                             -------------------
    begin                : Wed Apr 16 2003
    copyright            : (C) 2003 by
    email                :  eeyurcim@nottingham.ac.uk
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
 //this code has been put to gether by hacking through pic2pic.c
 again thanks to 2.10.95, rolf bippus
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "usralloc.h"
#include "ascio.h"
#include "fname.h"
#include "argio.h"
#include "piclib.h"
#include "cmdlnerr.h"
#include "image.h"

short houtpic;

void save_data_tiff(char* file,struct windows* win,int sq_width,int sq_height,char* label)
{
PICCREATESTRUCT pcstr;
unsigned char ** result;
unsigned short bytewidth=0L;
win->error_in_output=0;
TClear (&pcstr, 1);           //dont understand the point for this line
    pcstr.xdim = win->windows_x*sq_width;
    pcstr.ydim = win->windows_y*4*sq_height;
    pcstr.bits_per_pixel = 8;         //I guesed this value
    pcstr.file_type   = PIC_UNDEFINED;  /* kein Defaulttyp bei unbekannter oder
                                           keiner Extension */
    pcstr.color_type  = PIC_GRAYLEVEL;//PicColor(hinpic);
    pcstr.form_type   = PIC_PIXEL;
    pcstr.orientation = PIC_OR_LIOB;        //ref p 17 book 2   PicOrient
    pcstr.fill_order = PIC_UNDEFINED;

    pcstr.photometric = PIC_UNDEFINED; /* zunaechst undefiniert, nach Datei-
                                          osffnung abfragen */

    /* Bildbeschreibung enthaelt alle Parameter der Verarbeitung */
    pcstr.pic_contents = label;
    /* Software enthaelt die Commandozeile */
    pcstr.software = "";
    /* Source enthaelt den Namen der Eingangsdatei */
    pcstr.pic_source = file;
    /* Copy Right */
    pcstr.copy_right = "Produced in IFN Braunschweig";



  if ( (houtpic = PicOpen(file,
                          PIC_WRITE,
                          PIC_TIFF,
                          &pcstr)                    ) < 0 )
      CmdExitMess(CMDLN_OUTPUTOPEN,
                  "\nFehler beim oeffnen von %s zum schreiben:\n%s",
                  file, PicError(PIC_LAST_ERR) );
```

```
   if ( PicEmptyPixPic ( &pcstr, &result, 0, &bytewidth ) < 0 )           //this byte
width seems a little odd
      CmdExit (CMDLN_NOMEM);

  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {     // y  x
      fill_rect(x*sq_width,(y*4)*sq_height,sq_width,sq_height,result,double_to_char(win-
>win[win->windows_x-x-1][y].dat.north,win));
      fill_rect(x*sq_width,
((y*4)+1)*sq_height,sq_width,sq_height,result,double_to_char(win->win[win->windows_x-x-1]
[y].dat.east,win));
      fill_rect(x*sq_width,
((y*4)+2)*sq_height,sq_width,sq_height,result,double_to_char(win->win[win->windows_x-x-1]
[y].dat.south,win));
      fill_rect(x*sq_width,
((y*4)+3)*sq_height,sq_width,sq_height,result,double_to_char(win->win[win->windows_x-x-1]
[y].dat.west,win));

    }
  }

//fill_rect(4,4,20,result,201);



          if ( PicPutPixPic  ( houtpic, result[0] ) < 0 )
      CmdExit (CMDLN_PICLIBWRITE);
           PicClose ( houtpic );
}

void fill_rect(int in_x,int in_y,int size_x,int size_y,unsigned char **array,unsigned
char val)
{
int x=0;
int y=0;
    for (x=in_x;x<in_x+size_x;x++)
    for (y=in_y;y<in_y+size_y;y++)
    {
      array[y][x]=val;
    }

}

unsigned char double_to_char(double input,struct windows* win)
{
  if (input>0xFF)
  {
  win->error_in_output+=input-255;       //make a sum of the total error in the picture
  return 255;
  }
return (unsigned char)input;
}


void save_gnu_plot_skpt_file(char* filename_data,char* filename,struct windows* win,int
output_window_height)
{
FILE *config;
int x=0;
int y=0;
    if (!(config = fopen ( filename, "wb" )))
    {
      printf("file not opened");
    }
    fprintf (config, "set terminal png\nset output 'out.png'\n");
    fprintf (config, "set yrange [0:%i] reverse\n",output_window_height);
    for (y=0;y<win->windows_y;y++)
```

```
    for (x=0;x<win->windows_x;x++)
    {

      fprintf (config, "set label 'n%.0f' at %i,%i center font \"Courier,20\" \n",win-
>win[x][y].dat.north,win->win[x][y].pos_x,win->win[x][y].pos_y+2);
      fprintf (config, "set label 'e%.0f' at %i,%i center font \"Courier,20\" \n",win-
>win[x][y].dat.east,win->win[x][y].pos_x,win->win[x][y].pos_y+4);
      fprintf (config, "set label 's%.0f' at %i,%i center font \"Courier,20\" \n",win-
>win[x][y].dat.south,win->win[x][y].pos_x,win->win[x][y].pos_y+6);
      fprintf (config, "set label 'w%.0f' at %i,%i center font \"Courier,20\" \n",win-
>win[x][y].dat.west,win->win[x][y].pos_x,win->win[x][y].pos_y+8);

    }

    char filename_local[255];
    extract_file_name_from_path(filename_data,filename_local);

    fprintf (config, "plot '%s' wi li\n",filename_local);
    fclose(config);

}

void make_script_gnuplot_to_pmg(char* filename,char* dat)
{
FILE *config;
char temp[255];
char temp2[255];
    extract_file_name_from_path(dat,temp2);

    strcpy(temp,filename);
    strcat(temp,".bsh");
    if (!(config = fopen ( temp, "wb" )))
    {
      printf("file not opened");
    }
    fprintf (config, "gnuplot %s >out.png\n",temp2);
    fprintf (config, "kpaint out.png");
    fclose(config);

}

void extract_file_name_from_path(char* in,char* out)
{
int i=0;
int ii=0;
int iii=0;
strcpy(out,in);
for (i=strlen(in);i>0;--i)
{
  if (in[i]=='/')
  {
    for (ii=i+1;ii<strlen(in);ii++)
    {

      out[iii++]=in[ii];
    }
      out[iii++]='\0';
      break;
  }
}
//strcpy(in,temp);
}
```

7.1.7 norm.c

```
/***************************************************************************
                          norm.c  -  description
                             -------------------
    begin                : Thu Apr 17 2003
    copyright            : (C) 2003 by
```

```
    email                : eeyurcim@nottingham.ac.uk
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
#include <stdio.h>
#include <math.h>
#include "image.h"



void norm_data_to_window_hight(struct windows* win)
{
  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    win->win[x][y].dat.north/=win->win[x][y].size_y;
    win->win[x][y].dat.east/=win->win[x][y].size_y;
    win->win[x][y].dat.south/=win->win[x][y].size_y;
    win->win[x][y].dat.west/=win->win[x][y].size_y;
    }
  }

}

//why do I have two of these functions with difrent names
void norm_data_to_redy_for_image_transfer(struct windows* win)
{
  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    win->win[x][y].dat.north/=win->win[x][y].size_y;
    win->win[x][y].dat.east/=win->win[x][y].size_y;
    win->win[x][y].dat.south/=win->win[x][y].size_y;
    win->win[x][y].dat.west/=win->win[x][y].size_y;
    }
  }

}

void get_avrage_of_directions(struct windows* win)
{
  int x=0;
  int y=0;
  double tot_n=0;
  double tot_e=0;
  double tot_s=0;
  double tot_w=0;


  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    tot_n+=win->win[x][y].dat.north;
    tot_e+=win->win[x][y].dat.east;
    tot_s+=win->win[x][y].dat.south;
```

```
    tot_w+=win->win[x][y].dat.west;
    }
  }

tot_n/=(win->windows_x+1)*(win->windows_y+1);        //the one is to take care of the 0
posed data in the c array
tot_e/=(win->windows_x+1)*(win->windows_y+1);        //the one is to take care of the 0
posed data in the c array
tot_s/=(win->windows_x+1)*(win->windows_y+1);        //the one is to take care of the 0
posed data in the c array
tot_w/=(win->windows_x+1)*(win->windows_y+1);        //the one is to take care of the 0
posed data in the c array

win->stat_n.avg=tot_n;
win->stat_e.avg=tot_e;
win->stat_s.avg=tot_s;
win->stat_w.avg=tot_w;
}

void get_std_of_data(struct windows* win)
{

  int x=0;
  int y=0;
  double tot_n=0;
  double tot_e=0;
  double tot_s=0;
  double tot_w=0;

  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
    tot_n+=pow((win->win[x][y].dat.north-win->stat_n.avg),2);
    tot_e+=pow((win->win[x][y].dat.east-win->stat_e.avg),2);
    tot_s+=pow((win->win[x][y].dat.south-win->stat_s.avg),2);
    tot_w+=pow((win->win[x][y].dat.west-win->stat_w.avg),2);
    }
  }
if (tot_n>0) tot_n/=sqrt((win->windows_x+1)*(win->windows_y+1)-1);        //the one is to
take care of the 0 posed data in the c array
if (tot_e>0) tot_e/=sqrt((win->windows_x+1)*(win->windows_y+1)-1);        //the one is to
take care of the 0 posed data in the c array
if (tot_s>0) tot_s/=sqrt((win->windows_x+1)*(win->windows_y+1)-1);        //the one is to
take care of the 0 posed data in the c array
if (tot_w>0) tot_w/=sqrt((win->windows_x+1)*(win->windows_y+1)-1);        //the one is to
take care of the 0 posed data in the c array

win->stat_n.std=tot_n;
win->stat_e.std=tot_e;
win->stat_s.std=tot_s;
win->stat_w.std=tot_w;
}

void get_min_value(struct windows* win)
{
win->stat_n.min=0;        //and store back in the class
win->stat_e.min=0;
win->stat_s.min=0;
win->stat_w.min=0;
               //-val dose not happen and there is allways going to be an empty square
}

void get_max_value(struct windows* win,struct global_stats_and_errors *errors)
{
/*  int x=0;
  int y=0;
  double max_n=0;
  double max_e=0;
  double max_s=0;
  double max_w=0;
```

```
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {
      //minus dose not happen so dont wory about it
      if (win->win[x][y].dat.north>max_n) max_n=win->win[x][y].dat.north;
      if (win->win[x][y].dat.east>max_e) max_e=win->win[x][y].dat.east;
      if (win->win[x][y].dat.south>max_s) max_s=win->win[x][y].dat.south;
      if (win->win[x][y].dat.west>max_w) max_w=win->win[x][y].dat.west;

    }
  }
//store away the values for the errors analasis
if (errors->max_n<max_n) errors->max_n=max_n;
if (errors->max_e<max_e) errors->max_e=max_e;
if (errors->max_s<max_s) errors->max_s=max_s;
if (errors->max_w<max_w) errors->max_w=max_w;
           */
//data extracted from the abopuve functions
win->stat_n.max=1.571429;//max_n;      //and store back in the clas
win->stat_e.max=0.60;//max_e;
win->stat_s.max=1.391304;//max_s;
win->stat_w.max=0.733333;//max_w;

//win->stat_n.max=1.26;

//return 1.26;        //hard set the max value found from experiments
}

void init_stats(struct windows* win,double V,struct global_stats_and_errors *errors)
{
get_max_value(win,errors);
get_min_value(win);
get_avrage_of_directions(win);
get_std_of_data(win);     //nb the avrage must be found before the std is
w_max(win,V);
w_min(win,V);
}

void w_min(struct windows* win,double V)
{
/*if (win->stat.min<win->stat.avg-V*win->stat.std)
{
return win->stat.avg-V*win->stat.std;
}

return win->stat.min;
*/
if (win->stat_n.min<win->stat_n.avg-V*win->stat_n.std)
{
win->stat_n.w_min = win->stat_n.avg-V*win->stat_n.std;
}else
{
win->stat_n.w_min=win->stat_n.min;
}

if (win->stat_e.min<win->stat_e.avg-V*win->stat_e.std)
{
win->stat_e.w_min = win->stat_e.avg-V*win->stat_e.std;
}else
{
win->stat_e.w_min=win->stat_e.min;
}

if (win->stat_s.min<win->stat_s.avg-V*win->stat_s.std)
{
win->stat_s.w_min = win->stat_s.avg-V*win->stat_s.std;
}else
{
win->stat_s.w_min=win->stat_s.min;
```

```
}

if (win->stat_w.min<win->stat_w.avg-V*win->stat_w.std)
{
win->stat_w.w_min = win->stat_w.avg-V*win->stat_w.std;
}else
{
win->stat_w.w_min=win->stat_w.min;
}

}

void w_max(struct windows* win,double V)
{

if (win->stat_n.max>win->stat_n.avg+V*win->stat_n.std)
{
win->stat_n.w_max =win->stat_n.avg+V*win->stat_n.std;
}else
{
win->stat_n.w_max=win->stat_n.max;
}

if (win->stat_e.max>win->stat_e.avg+V*win->stat_e.std)
{
win->stat_e.w_max =win->stat_e.avg+V*win->stat_e.std;
}else
{
win->stat_e.w_max=win->stat_e.max;
}

if (win->stat_s.max>win->stat_s.avg+V*win->stat_s.std)
{
win->stat_s.w_max =win->stat_s.avg+V*win->stat_s.std;
}else
{
win->stat_s.w_max=win->stat_s.max;
}

if (win->stat_w.max>win->stat_w.avg+V*win->stat_w.std)
{
win->stat_w.w_max =win->stat_w.avg+V*win->stat_w.std;
}else
{
win->stat_w.w_max=win->stat_w.max;
}

}

double x_norm_n(struct windows* win,double in)
{
  if (in<win->stat_n.w_min)
  {
  return win->stat_n.w_min;
  }

  if (in>win->stat_n.w_max)
  {
  return win->stat_n.w_max;
  }

  //else
  return in;

}


double x_norm_e(struct windows* win,double in)
{
  if (in<win->stat_e.w_min)
  {
  return win->stat_e.w_min;
```

```
  }

  if (in>win->stat_e.w_max)
  {
  return win->stat_e.w_max;
  }

  //else
  return in;

}

double x_norm_s(struct windows* win,double in)
{
  if (in<win->stat_s.w_min)
  {
  return win->stat_s.w_min;
  }

  if (in>win->stat_s.w_max)
  {
  return win->stat_s.w_max;
  }

  //else
  return in;

}

double x_norm_w(struct windows* win,double in)
{
  if (in<win->stat_w.w_min)
  {
  return win->stat_w.w_min;
  }

  if (in>win->stat_w.w_max)
  {
  return win->stat_w.w_max;
  }

  //else
  return in;

}


double normme_n(struct windows* win,double in,double max)
{
double out=0;
if (win->stat_n.w_max-win->stat_n.w_min)            //devide by 0 protection
{
out= ((x_norm_n(win,in)-win->stat_n.w_min)/(win->stat_n.w_max-win->stat_n.w_min))*max;
}else
{
printf("win->stat.w_max-win->stat.w_max has produce a devide by 0 oops.\n");
}
  return out;
}

double normme_e(struct windows* win,double in,double max)
{
double out=0;
if (win->stat_e.w_max-win->stat_e.w_min)            //devide by 0 protection
{
out= ((x_norm_e(win,in)-win->stat_e.w_min)/(win->stat_e.w_max-win->stat_e.w_min))*max;
}else
{
printf("win->stat.w_max-win->stat.w_max has produce a devide by 0 oops.\n");
}
  return out;
}
```

```
double normme_w(struct windows* win,double in,double max)
{
double out=0;
if (win->stat_w.w_max-win->stat_w.w_min)            //devide by 0 protection
{
out= ((x_norm_w(win,in)-win->stat_w.w_min)/(win->stat_w.w_max-win->stat_w.w_min))*max;
}else
{
printf("win->stat.w_max-win->stat.w_max has produce a devide by 0 oops.\n");
}
  return out;
}

double normme_s(struct windows* win,double in,double max)
{
double out=0;
if (win->stat_s.w_max-win->stat_s.w_min)            //devide by 0 protection
{
out= ((x_norm_s(win,in)-win->stat_s.w_min)/(win->stat_s.w_max-win->stat_s.w_min))*max;
}else
{
printf("win->stat.w_max-win->stat.w_max has produce a devide by 0 oops.\n");
}
  return out;
}

void norm_data_set(struct windows* win)
{

  int x=0;
  int y=0;
  if (win->win) //if we have our mem
  {
    for (x=0;x<win->windows_x;x++)
    for (y=0;y<win->windows_y;y++)
    {

      //minus dose not happen so dont wory about it
      win->win[x][y].dat.north=normme_n(win,win->win[x][y].dat.north,255);
      win->win[x][y].dat.east=normme_e(win,win->win[x][y].dat.east,255);
      win->win[x][y].dat.south=normme_s(win,win->win[x][y].dat.south,255);
      win->win[x][y].dat.west=normme_w(win,win->win[x][y].dat.west,255);


    }
  }


}
```

Clustering Pearl Scripts

7.2.1 Server.pl

```
#!/usr/bin/perl

print "Command spreader copyright Roderick MacKenzie 2003\n";
print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";
print "Warning TCP command interface open\n";
print "Warning HTTP interface open\n";
print "idleing...\n";

use IO::Socket;
use Getopt::Long;

@commands=();
$ncommands=0;
$command="";
```

```
$enabled="false";
$password="";
$root="./webpages";
$freinds="";
$task=0;
$port=1250;
$encryption_key="This is an encryption key";
@net_commands=("")x100;
@cluster_computers=();
@cluster_computer_ports=();
$ncluster_computers=0;
$refresh_rate=100;
sub reset_globals
{
    $#commands=0;
    $ncommands=0;

}

sub replace
{
($text,$to_replace,$replace_with) = @_;
substr($text,index($text,$to_replace),length($to_replace),$replace_with);
return $text;
}



sub can_display_page
{
($in,$ip)=@_;
#If the page is a plblic page
if (($in eq "/index.html")||
    ($in eq "/")||
    ($in eq "/index-1.html")||
    ($in eq "/index-2.html")||
    ($in eq "/index-5.html")||
    ($in eq "/index-6.html")
    )

{return 1;}

#If the page is a private page
if (($in eq "/index-3.html")||
    ($in eq "/index-4.html"))
    {
             if (index($freinds,$ip)!=-1)
             {
             return 1;                #The page is alowed
             }else
             {
             return 2;                #The page is forbiden from this adress
             }
    }


return 0;    #if the page is out right not there or forbiden
}

sub loop_back
{
($command)=@_;

                                     if (defined($socket = IO::Socket::INET->new
                            (
                                PeerAddr => "localhost",
                                PeerPort => $port,
                                Proto    => "tcp",
                                Type     => SOCK_STREAM
                            ))){

                                         print $socket "child task",$socket->peerhost,"
```

```
$command";
close ($socket);}
}


sub tx_webpage
{
($client,$page)=@_;
             if (open(FILE, $page))
             {
             binmode(FILE);
             print $client "$httpver 200 OK\n\n";
             while (<FILE>){print $client $_;};
             close(FILE);
             }

}

sub web_server
{
($line,$client)=@_;
    @header = split(' ', $line);
    $url = $header[1];
    $httpver = $header[2];

    # if it was an valid header, begin
    if ($header[0] eq "GET" && ($httpver eq "HTTP/1.0" || $httpver eq "HTTP/1.1")) {

    if (can_display_page($url,$client->peerhost)==0)             #The page is not found
    {
             tx_webpage($client,$root."/error.html");
    }


    if (can_display_page($url,$client->peerhost)==2)             #The page is private and
forbiden
    {
             tx_webpage($client,$root."/forbiden.html");
    }

    if (can_display_page($url,$client->peerhost)==1)             #The page is alowed
display it
    {

               # if the document is a directory, append the defaultfile name (index.html)
                 if (rindex($url, "/") == length($url)-length("\n")) { $url.=
"index.html" }

                 if (open(FILE, $root.$url))
                 {
                 binmode(FILE);

                 # send an HTTP header
                 print $client "$httpver 200 OK\n\n";

                 # send the file
                 while ($in=<FILE>)
                 {
                          if (index($in,"KEY_STATUS_LOG")!=-1)
                          {
                              for ($i=0;$i<$ncluster_computers;$i++)
                              {

$name=substr(@cluster_computers[$i],0,length(@cluster_computers[$i])-
1).":".substr(@cluster_computer_ports[$i],0,length(@cluster_computer_ports[$i])-
1)."/index-4.html";
                                        print $client "\n<P><A HREF=\"http://
$name\">$name</A></P>\n";
                                        print @cluster_computers[i];
                              }
                          print $client @net_commands;
```

```
                        }
                        else
                        {
                        if ((index($in,"KEY_CPU_INFO")!=-1))
                        {
                            if (open(han, "/proc/cpuinfo"))
                            {
                                    while (<han>)
                                    {print $client "<P>$_\n</P>";
                                    }
                                close (han);
                            }
                        }else
                        {
                            print $client $in;
                        }
                        }
                }

                print $client $cpuinfo;
                close(FILE);

                # print log
                print "[", $client->peerhost, "] ", ctime, " send \"$url\"\n";
                }else
                {
                        tx_webpage($client,$root."/error.html");
                }

    }

close $client;
}

}

GetOptions("port=s" => \$set_port,
                "refresh=s" => \$refresh,
               "key=s" => \$enc_key);

if ($set_port)
{
$port=$set_port;
}

if ($refresh)
{
$refresh_rate=$refresh;
print "Refresh rate: $refresh_rate\n";
}

if ($enc_key)
{
$encryption_key=$enc_key;
}

print "Using port $port\n";

$maxcon=SOMAXCONN;
print "max conections to server $maxcon\n";
$server = IO::Socket::INET->new
(
    LocalPort => $port,
    Type      => SOCK_STREAM,
    Reuse     => 1,
    Listen    => $maxcon
) or die "Could not create server.\n";

while ($client = $server->accept())
{

#    unless (defined($child_pid = fork())) {die "Can not fork.\n"};
```

```
#    if ($child_pid) {
             #$password=<$client>;                                  #the first data that is
sent to me is my password
             #print "just before reset  $password";
             RESET: while ($line = <$client>)
             {

            if ($password eq "")
                 {
                          $password = $line;
                          print "password set to $password\n";
                          $freinds=$client->peerhost;                          #I make
frinds with the first person I meat.
                 #print $client "Hi I am there\nHi \n Hi\n";
                          $line="system now secure\n";
                 }



            print $client->peerhost,": $line";
                 web_server($line,$client);


                 if (index($line,"child task")==0)
                 {
                          substr($line,0,10,"");    #Extract the task fromt the child
                          unshift (@net_commands,"<P>$line</P>");
                          pop  @net_commands;
                 }



                 if (($enabled eq "true") && (index($freinds,$client->peerhost)!=-1))
                 {
             #if we get this far the code is enabled and redy to run

                          if ($line eq "killkill\n")
                          {
                              print "kill command recived";
                              exit;                                            #exit
nicley
                          }


                          if ($line eq "list_mode\n")
                          {
                              $line =<$client>;
                              $file = "./temp".$task.$line;                    #the next
input will be the file name
                              print "Saveing List\n";
                              if (open(han,">$file"))
                              {

                                       print han @commands;

                                  close (han);
                              }
                              print "List saved\n";              #now that the list has
been saved  to disk clear it
                              $ncommands=1;
                              $#commands=0;

                              $line =<$client>;
                              $line = replace ($line,"<list>",$file);
                              push (@commands,$line);
                              $ncommands++;
                          }


                          if ($line eq "load_list\n")
                          {
```

```
                        print "Loading list\n";
                            LIST_LOAD: while ($coms = <$client>)
                            {
                                        if ($coms eq "exit_list\n")
                                        {
                                                $ncommands++;
                #I dont know why this is needed, but it seems to need it
                                                print "exit_list called\n";
                                            last LIST_LOAD;
                                        }

                                        push (@commands,$coms);
                                        $ncommands++;

                                        if ($coms eq "resetreset\n")
                                        {
                                                $ncommands=0;
                                                $#commands =0;
                                                last LIST_LOAD;
                                        }


                            }
                            #print @commands;
                            print "finidhed loading command list\n";
                        }

                        if ($line eq "update_cluster_list\n")
                        {
                        print "Loading cluster list\n";
                     $ncluster_computers=0;                      #flush old list
                        $#cluster_computers=-1;
                        $#cluster_computer_ports=-1;

                            LIST_LOAD_CLSTER: while ($coms = <$client>)
                            {
                                        if ($coms eq "exit_list\n")
                                        {
                                                #$ncluster_computers++;
                #I dont know why this is needed, but it seems to need it
                                                print "exit_list called\n";
                                            last LIST_LOAD_CLSTER;
                                        }
                                        push (@cluster_computers,"$coms");
                                        $portinput=<$client>;
                                        push (@cluster_computer_ports,$portinput);
                                        $ncluster_computers++;

                                        if ($coms eq "resetreset\n")
                                        {
                                                reset_globals();
                                                last LIST_LOAD_CLUSTER;
                                        }


                            }
                            print "finidhed loading list of cluster computers
$ncluster_computers\n";
                        }


                        if ($line eq "resetreset\n")
                        {
                                    reset_globals();
                        }


                        if ($line eq "run commands now\n")
                        {
                                    close ($client);
                #kill the conection now!
                                    $enabled="false";                       #make shure
```

```
after the operation the input is disabled
                                $task++;                        #add one to the task
number
                                    unless (defined($command_child = fork())) {die
"Can not fork to produce command child.\n"};
                                    if ($command_child)
                                    {


                                        loop_back("Job $task started $ncommands to
process");


                                        print "running commands in command child\n";
                                        @last_commands=(" "," "," "," "," "," "," ","
");
                                        $c=0;
                                        for ($i=0;$i<$ncommands;$i++)
                                        {
                                                #for ($i=0;$i<$len_index;$i++)
                                                #{
                                                #$work_arg=$command;
                                                #print
replace($work_arg,"<list>",@commands[$i]);
                                                #}
                                                #print "Next command isued
@commands[$i]\n";
                                                system(@commands[$i]);
                                                #print $last_command;
                                                if (($c++)==$refresh_rate)
                                                {
                                                $c=0;
                                                $done=($i/$ncommands)*100;
                                                loop_back( "Job ".$task." ".$i."of".
$ncommands." commands done ".$done."%  last command was:".@commands[$i]);
                                                }
                                        }

                                        loop_back("Job $task done 100%");
                                        print "Terminating child\n";
                                        exit $command_child;              #kill the
child
                                    }
                            print "Server Still Running.. state:idleing\n";
                            }
#
                            }else
                    {                       #wait for a the enabling key
                            $encoded = (crypt $password ,$encryption_key);
                            $encoded .= "\n";
                            if ($line eq $encoded)
                            {
                                print "password accepted active for $friends\n";
                                $enabled = "true";
                            }
                    }   #       if ($enabled eq "true")


           }                #       RESET: while ($line = <$client>)
#       } else               #other process begins
#       {
#                   while ($line = <$client>)
#                   {
#                           print "Sending command through secondry parser $line";
#                           if  ($line eq "stopstop\n")
#                           {
#                               print "Stopping exercution\n";
#                               $stop=1;
#                           }
#                   }
#       }
}
```

```
close ($client);
exit;
```

7.2.2 Client.pl

```
#!/usr/bin/perl
use IO::Socket;
use Getopt::Long;
#$inputfile="index";              #input file
$computers=4;                    #numberof computers in a system
$password="This is my secure password";
$encryption_key="This is an encryption key";
@computer_name = ("sbv9","sbv11","sbv5","sbv12");
@port                     = (1250,1250,1250,1250);
@index                    = ();
$len_index   =0;
##################################################################
#
#            Functions defined below
#
#
##################################################################


sub replace
{
($text,$to_replace,$replace_with) = @_;
substr($text,index($text,$to_replace),length($to_replace),$replace_with);
return $text;
}

sub read_in_data
{
#read in the data
open (filehan,"<$inputfile") or die ("Can not open file");
$len_index=0;
while ($text=<filehan>){  #read(filehan,$text,1)){
    push (@index,$text);
    $len_index++;
#    print $text;
}
}

sub enable_computer
{
    ($computer,$pword,$name)=@_;
    $pword= crypt $pword, $name;

    $encoded =crypt $pword, $encryption_key;
    $encoded .= "\n";
    printf  $computer  $encoded;
    print $encoded;
}

sub password_computers
{
    ($pword)=@_;
    for ($cur_computer=0;$cur_computer<$computers;$cur_computer++)
    {


    $socket = IO::Socket::INET->new
    (
```

```
            PeerAddr => @computer_name[$cur_computer],
            PeerPort => @port[$cur_computer],
            Proto    => "tcp",
            Type     => SOCK_STREAM
    ) or die "Could not create client during passwording stage.\n";
            $temp= crypt $pword, @computer_name[$cur_computer];
            $temp .="/n";
        printf  $socket $temp;

    close ($slcket);

    }
}

sub help
{
    print "usage:\n perl spread \"command <list> arguments\"\n";
}

sub tx_cluster_names_and_adresses()
{
            for ($cur_computer=0;$cur_computer<$computers;$cur_computer++)
            #run through the list of computers
            {
                $socket = IO::Socket::INET->new
                (
                    PeerAddr => @computer_name[$cur_computer],
                         PeerPort => @port[$cur_computer],
                         Proto    => "tcp",
                         Type     => SOCK_STREAM
                ) or die "Could not create client.\n";

                enable_computer($socket,$password,@computer_name[$cur_computer]);

                printf  $socket  "update_cluster_list\n";
                         for ($i=0;$i<$computers;$i++)                      #run
through the list of computers
                         {
                             printf  $socket  "@computer_name[$i]\n";
                             printf  $socket  "@port[$i]\n";

                         }

                printf  $socket  "exit_list\n";

                close ($socket);
            }


}

#########################################################################
#
#The begining of the main loops and code
#
#
#
#########################################################################

#getopt('lk');
GetOptions("list" => \$list_mode,
                "command=s" => \$load_mode,
                "kill" => \$kill_mode,
                "key=s" => \$enc_key,
                "file=s" => \$inputfile);


if ($enc_key)
{
$encryption_key=$enc_key;
}
```

```
#unless (defined($child_pid = fork()))
#    {
#            die "Can not fork.\n"
#    };

#if ($child_pid)
#{

    print "\nDistrubuting command through \n";
    for ($cur_computer=0;$cur_computer<$computers;$cur_computer++)
    {
    print "@computer_name[$cur_computer] @  @port[$cur_computer] \n";
    }

    if ($list_mode)
    {
    print "List mode enabled\n";
    }

    $password=crypt $password, "35jhb3hj45vfh3c5j3hbk45";
             #encode the password
    password_computers($password);                  #password the computers

    #they should all now be in a secure and disabled state

    if ($kill_mode)
    {
             print "Do you realy want to remove all loaded servers?(Y/N):";
             $in=<>;
             if ($in eq "Y\n")
             {
             for ($cur_computer=0;$cur_computer<$computers;$cur_computer++)
             {
                 $socket = IO::Socket::INET->new
                 (
                 PeerAddr => @computer_name[$cur_computer],
                 PeerPort => @port[$cur_computer],
                 Proto    => "tcp",
                 Type     => SOCK_STREAM
                 ) or die "Could not create client.\n";
                 enable_computer($socket,$password,@computer_name[$cur_computer]);
                 print "removing client from: @computer_name[$cur_computer]\n";
                 printf  $socket  "killkill\n";
                 close ($socket);


             }
             }else{print "not killing"}
    exit;
    }

    unless ($inputfile)
    {
    help;
    die "no input file defined";
    }


    if (index($load_mode,"<list>") == -1)
    {
             help;
             die "<list> opperator not found in 0th argument";
    }

    #by this stage in program we know that we are going to do some usefull work
    #tell the clients who there nabhors are
    tx_cluster_names_and_adresses();



    read_in_data;                       #Reads in the data
    unless ($list_mode)
```

```
    {
    for ($i=0;$i<$len_index;$i++)
    {
    $argtemp=$load_mode;
             #unless ($list_mode)      #if we are opperating in on the fly mode
             #{
                 @index[$i] =
replace($argtemp,"<list>",substr(@index[$i],0,length(@index[$i])-1));
                 @index[$i] .= "\n";
             #}#else                   # if we need to compile a list first
             #{
             #    @index[$i] = replace($argtemp,"<list>",$list_mode);
             #place the name of the temp file in the output
             #    @index[$i] .= "\n";
             #}
    }
    }

    #print @index;

    #print @index;
    #exit;

    # split up the data
    $data_per_computer=$len_index/$computers;

    for ($cur_computer=0;$cur_computer<$computers-1;$cur_computer++)
    {


    $socket = IO::Socket::INET->new
    (
        PeerAddr => @computer_name[$cur_computer],
        PeerPort => @port[$cur_computer],
        Proto    => "tcp",
        Type     => SOCK_STREAM
    ) or die "Could not create client.\n";
             enable_computer($socket,$password,@computer_name[$cur_computer]);
        printf  $socket  "resetreset\n";
        printf  $socket  "load_list\n";



    for ($i=$data_per_computer*$cur_computer;$i<($data_per_computer*$cur_computer)+
$data_per_computer;$i++)
    {
        printf  $socket  @index[$i];

             #print $cur_computer;
    }



        printf  $socket  "exit_list\n";

        if ($list_mode)
             {
                         printf  $socket  "list_mode\n";
                         printf  $socket  @computer_name[$cur_computer]."\n";
                         printf  $socket  "$load_mode\n";

             }

             printf  $socket "run commands now\n";
             close ($socket);

    };

    if ($computers==1)
    {
             $i=0;                                  # if the number of computers is 0
then make shure that the list starts at the begining
```

```
    }
    #print "the cu computer is $cur_computer";
    $#test=0;
    $socket = IO::Socket::INET->new
    (
        PeerAddr => @computer_name[$cur_computer],
        PeerPort => @port[$cur_computer],
        Proto    => "tcp",
        Type     => SOCK_STREAM
    ) or die "Could not create client.\n";

             enable_computer($socket,$password,@computer_name[$cur_computer]);
            printf  $socket  "resetreset\n";
            printf  $socket  "load_list\n";


    for ($i;$i<$len_index;$i++)
    {
        printf  $socket @index[$i];
#            print $cur_computer;
    }
        printf  $socket  "exit_list\n";

        if ($list_mode)
             {
             printf  $socket  "list_mode\n";
             printf  $socket  @computer_name[$cur_computer]."\n";
             #@index[$i] =
replace($argtemp,"<list>",substr(@index[$i],0,length(@index[$i])-1));
             printf  $socket  "$load_mode\n";
             }

             printf $socket            "run commands now\n";

    close ($socket);


    close (filehan);

#}else
#{
#    while($hi = <$socket>)
#    {
#    print "Read this from server: $hi";
#    }
#
#}


#unless (defined($child_pid = fork()))
#    {
#            die "Can not fork.\n"
#    };

#if ($child_pid) {

#    while ($line = <>) {
#
#        print $socket $line;
#
#    }

#} else {

#    while($line = <$socket>) {
#
#        print "Read this from server: $line";
#
#    }

#}
#             txtdata $socket;
```

```
#}                                      #end of for loop
```

7.3.1 Linux cluster - server.pl

```
#include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/sysinfo.h>
#include <unistd.h>
#include <resolv.h>
#include <arpa/inet.h>
#include <errno.h>
#include "vars.h"
#include <sys/resource.h>
#include <signal.h>
#include <stdlib.h>


#include <stdarg.h>

#include <string.h>

#define MY_PORT                         1250
#define MAXBUF                          1024
#define max_jobs          100
#define alarm_time        5
#define true              1
#define false                           0
#define RUN_LOAD          140000

void PANIC(char *msg);
#define PANIC(msg)  {perror(msg); abort();}


int send_message(struct my_job* job);
char my_name[32];

struct my_job jobs[max_jobs];                                                   //we can
have 100 jobs in the list
/* This flag controls termination of the main loop. */
//volatile sig_atomic_t keep_going = 1;

/* The signal handler just clears the flag and re-enables itself. */
void MakeHex(char* store,char data);

void print_wild_chars(char *data,int len)
{
int i=0;
char store[4];
printf("len now is%i \n\n\n\n\n\n\n",len);
             for (i=0;i<len;i++)
             {
             MakeHex(store,data[i]);
             printf("%s",store);
             }
}



void MakeHex(char* store,char data)
{
```

```
                  if ((data==45)||(data==32)||(data=='.')||(data==35)||(data>='a')
                  &&(data<='z')||(data>='A')&&(data<='Z')||(data>='0')&&(data<='9'))
                  {
                    sprintf(store,"%c",data);
                          }
//                        else
//                        {
//                        if (data) sprintf(store,"[%x]",data);
//                        }
}



int can_i_run_a_task()
{
//struct rusage me;
//struct rlimit that;
//struct timeval tv;
//struct timezone zone={0,0};
//double me_time=0;
//           gettimeofday(&tv,&zone);
//           getrlimit(RLIMIT_DATA,&that);
//           getrusage(RUSAGE_SELF,&me);
//           me_time=(double)me.ru_utime.tv_usec+(double)me.ru_stime.tv_usec + ((double)
(me.ru_utime.tv_sec+me.ru_stime.tv_sec )*100000000);
//                   printf("signal1s %f\n",me_time);
struct sysinfo info;
sysinfo(&info);
printf ("load %d %i \n",info.loads[0],RUN_LOAD);

if (info.loads[0]<RUN_LOAD)
{
    return true;
}else
{
    return false;
}

}

void clear_all_jobs(struct my_job* list)
{
int i=0;
    for (i=0;i<max_jobs;i++)
    {
             list[i].status='e';
    }
}

int get_next_job(struct my_job* list,char* my_name)
{
int i=0;
int found =-1;
    for (i=0;i<max_jobs;i++)
    {
             if (list[i].status!='e')                          //the job must have a
name ie be there
             if (strcmp(list[i].node,my_name)==0)              //the job must belong to
me
             if (list[i].status=='w')                          //only start job waiting
to be started
             {
             return i;    //break out of the function earley we have found a job
             }
    }

return found;
}

int get_number_of_jobs(struct my_job* list)
{
int i=0;
```

```
int found=0;
    for (i=0;i<max_jobs;i++)
    {
              if (list[i].status!='e')                              //the job must have a
name ie be there
              {
              found++;
              }
    }

return found;
}


int get_job_id(struct my_job* list,char *id)
{
int i=0;
int found =-1;
    for (i=0;i<max_jobs;i++)
    {
              if (strcmp(list[i].id,id)==0)                                     //the job
must have a name ie be there
              {
              found=i;
              }
    }

return found;
}


//this is an update command and an add command if the ID is allredy there then the item
is updated
void add_job(char* id,char* path,char* command,char* node,struct my_job* list,char*
came_from,char status,int pid)
{
int i=0;
int pos=get_job_id(list,id);

              if (pos==-1)              //if it is not found find a gap and add the job in
              {
                  for (i=0;i<max_jobs;i++)
                  {
                           if (list[i].status=='e')              //ie the list is empty
                           {
                               strcpy(list[i].id,id);
                               strcpy(list[i].path,path);
                               strcpy(list[i].command,command);
                               strcpy(list[i].node,node);
                               strcpy(list[i].came_from,came_from);
                               list[i].pid=pid;
                               list[i].status=status;
                               break;                                           //break the
for loop early
                           }
                  }

              }else                     //if we have found a job pos just update the job
info struct
              {
                  if(!
((list[pos].status=='r')&&(status=='w')&&(strcmp(list[pos].node,my_name)==0)))
              //If the job is running then nobody has the right to make it wait
                  {
                           if
((list[pos].status=='r')&&(status=='e')&&(strcmp(list[pos].node,my_name)==0))
              //ie we want to empty the job
                           {
                               printf("trying to kill %i\n",list[pos].pid);
                               kill(list[pos].pid,SIGKILL);
                           }
```

```
                        strcpy(list[pos].id,id);
                    strcpy(list[pos].path,path);
                    strcpy(list[pos].command,command);
                    strcpy(list[pos].node,node);
                        strcpy(list[pos].came_from,came_from);
                        list[pos].status=status;
                        list[pos].pid=pid;
                }
            }


}

void remove_job(struct my_job* list,char* id)
{
int i=0;
    for (i=0;i<max_jobs;i++)
    {
            if (strcmp(list[i].id,id)==0)                       //ie the list is empty
            {
                list[i].status='e';
                break;                                          //break the for loop
early
            }
    }

}

void debug_print_list(struct my_job* list)
{
int i=0;
    for (i=0;i<max_jobs;i++)
    {
            if (list[i].status!='e')              //ie the list is empty
            {
                printf("QPOS: %i %s %s %s %s %c
\n",i,list[i].node,list[i].path,list[i].command,list[i].id,list[i].status) ;
            }
    }
}

void catch_alarm (int sig)
{
//             keep_going = 0;
//int can_i_run_a_task=can_i_run_a_task();
int child=0;
int to_run=0;
int ret=0;

int stat;

signal (sig, catch_alarm);                                      //clear the signal
if (can_i_run_a_task()==true)
{
to_run=get_next_job(jobs,my_name);
//printf("I have found the next job as %i\n\n",to_run);
    if (to_run!=-1)                                 //if ther is a job left to run
    {
             jobs[to_run].status='r';                           //running
            if ((child=fork())==0)
            {
                        char* command;
                        command =(char*)calloc(strlen(jobs[to_run].path)
+strlen(jobs[to_run].command)+10,sizeof(char));     //the 10 is for luck
                        sprintf(command,"cd %s;
%s\n",jobs[to_run].path,jobs[to_run].command);
                        printf("I would have run:%s\n",command);

                        if ((ret=fork())==0)
                        {
                                    execlp("/bin/sh","sh","-c",command,0);
                                    //execvp(command,0,0);
```

```
                        }

                        strcpy(jobs[to_run].came_from,my_name);
            //set where we came from
                        jobs[to_run].pid=ret;
            //the +1 is because we are exercuting the command through anouther process
is sh
                        send_message(&jobs[to_run]);
//transmit back to the child the update of the PID

                        printf("I should be waiting on %i\n",ret);

                        waitpid(ret,&stat,0);                 //wait for the exit of
the fork
                        printf("I have finished with\n");
                        //ret=system(command);                //execeututed.
                        if (ret<0)
                        {
                        jobs[to_run].status='n';              //n for not worked
                        }else
                        {
                        jobs[to_run].status='f';              //finnished
                        }

                        strcpy(jobs[to_run].came_from,my_name);
            //set where we came from
                        send_message(&jobs[to_run]);
                        printf("updateing %s\n\n",jobs[to_run].id);
                        free(command);                                    //clean up
                        printf("Removing fork from mem\n");
                        exit(0);                              //kill fork I hope!
            }
//          jobs[to_run].pid=child;
    // remove_job(jobs,jobs[to_run].id);
     }
}else
{
printf("Load too high to run a task\n");
}

debug_print_list(jobs);
//send_message(&jobs[0]);

alarm(alarm_time);                                                        //set the
alarm off again
}

/*int send_message(struct my_job* job)              //sends a message in the form of a
job
{
    int sockfd, bytes_read;
    struct sockaddr_in dest;
    char buffer[MAXBUF];
    int i;
    if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
        PANIC("error creating endpoint coms descriptior");

    bzero(&dest, sizeof(dest));
    dest.sin_family = AF_INET;
    dest.sin_port = htons(1253);
    if ( inet_aton("localhost", &dest.sin_addr.s_addr) == 0 )
        PANIC("localhost");

    if ( connect(sockfd, (struct sockaddr*)&dest, sizeof(dest)) != 0 )
        PANIC("Connect");

    bzero(buffer, MAXBUF);

    strcpy(buffer,"load job");
    send(sockfd, buffer,100, 0);
```

```
    //struct my_job job;
    //strcpy(job->id,"hello world");
    //strcpy(job->node,"localhost");
    //strcpy(job->path," /root/adasd//adsas/");
    //strcpy(job->command,"find something");
    //job->status='f';

    send(sockfd, job,sizeof(struct my_job), 0);

    bzero(buffer, MAXBUF);
    strcpy(buffer,"close conection");
    send(sockfd, buffer,100, 0);



//    do
//    {
//        bzero(buffer, sizeof(buffer));
//        bytes_read = recv(sockfd, buffer, sizeof(buffer), 0);
//        if ( bytes_read > 0 )
//            printf("%s", buffer);
//    }
//    while ( bytes_read > 0 );


    close(sockfd);
}
*/

int send_message(struct my_job* job)
{
            struct sockaddr_in addr;
            struct sigaction act;
            int bytes;
            char buffer[MAXBUF];
//          bzero(&act, sizeof(act));
//          act.sa_handler = sig_handler;
//          act.sa_flags = SA_RESTART;
//          sigaction(SIGURG, &act, 0);
//          sigaction(SIGALRM, &act, 0);

int         serverfd = socket(PF_INET, SOCK_STREAM, 0);
//          /*---claim SIGIO/SIGURG signals---*/
//          if ( fcntl(serverfd, F_SETOWN, getpid()) != 0 )
//                      perror("Can't claim SIGURG and SIGIO");
            bzero(&addr, sizeof(addr));
            addr.sin_family = AF_INET;
            addr.sin_port = htons(MY_PORT);
            inet_aton(my_name, &addr.sin_addr);

            if ( connect(serverfd, (struct sockaddr*)&addr, sizeof(addr)) == 0 )
            {

                bzero(buffer, MAXBUF);
                strcpy(buffer,"load job");
                send(serverfd, buffer,100, 0);

            //struct my_job job;
            //strcpy(job->id,"hello world");
            //strcpy(job->node,"localhost");
            //strcpy(job->path," /root/adasd//adsas/");
            //strcpy(job->command,"find something");
            //job->status='f';
                send(serverfd, job,sizeof(struct my_job), 0);

                bzero(buffer, MAXBUF);
                strcpy(buffer,"close connection");
                send(serverfd, buffer,100, 0);


            }
```

```
            close(serverfd);


}



int main(int Count, char *Strings[])
{
//char my_name[30];                                                         //my name
char close_con=false;
clear_all_jobs(jobs);
char buffer[MAXBUF];
int send_jobs;
strcpy(my_name,"127.0.0.2");
char argv[2][20];
strcpy(argv[0],"ls -l");

//set my name
//add_job("0","root/hello/adasda/asdas","find ~/>work",my_name,jobs,'w');
//add_job("1","root/hello/adasda/asdas","find ~/>work",my_name,jobs,'w');
//add_job("2","root/hello/adasda/asdas","find ~/>work",my_name,jobs,'w');
//add_job("3","root/hello/adasda/asdas","find ~/>work",my_name,jobs,'w');
//remove_job(jobs,"fred blogs1");


               /* Establish a handler for SIGALRM signals. */
               signal (SIGALRM, catch_alarm);

                 /* Set an alarm to go off in a little while. */
                 alarm (alarm_time);

                   /* Check the flag once in a while to see when to quit. */


            struct sockaddr_in addr;
            int sd, addrlen = sizeof(addr);

            int i=0;                  //GPC
        struct my_job job;            //temp job storage var

            if ( (sd = socket(PF_INET, SOCK_STREAM, 0)) < 0 )
                        PANIC("Socket");
            addr.sin_family = AF_INET;
            addr.sin_port = htons(MY_PORT);
            addr.sin_addr.s_addr = INADDR_ANY;
            if ( bind(sd, (struct sockaddr*)&addr, addrlen) < 0 )
                        PANIC("Bind");
            if ( listen(sd, 20) < 0 )
                        PANIC("Listen");

            while (1)
            {           int rec;
                        int client = accept(sd, (struct sockaddr*)&addr, &addrlen);
                        printf("Connected: %s:%d\n", inet_ntoa(addr.sin_addr),
ntohs(addr.sin_port));

                        while((close_con==false)&&((rec = recv(client, buffer, 100,
MSG_WAITALL))!=0))

                        {
                            if (rec>0)
                            {
                                    printf ("I have recived something %s\n\n",buffer);
                                    if (strcmp(buffer,"exit now")==0)
                                    {
                                        printf("The exit command has been recived\n");
                                    }

                                    if (strcmp(buffer,"would you like a name")==0)
                        //recive a name
                                    {
```

```
                                        rec = recv(client, buffer, 100, MSG_WAITALL);
                                        strcpy(my_name,buffer);
                                        printf("My name is now %s\n",my_name);
                                    }

                                    if (strcmp(buffer,"load job")==0)
                                    {
                                        printf("Job update :\n");
                                        rec=recv(client, &job, sizeof(struct my_job),
MSG_WAITALL);
                                        printf("Job id %s\n", job.id);
                                        printf("Command %s\n", job.command);
                                        printf("Path %s\n", job.path);
                                        printf("node %s\n", job.node);
                                        printf("status %c\n", job.status);
//                                      print_wild_chars((char*)&job,sizeof(struct
my_job));


add_job(job.id,job.path,job.command,job.node,jobs,job.came_from,job.status,job.pid);
                                    }

                                    if (strcmp(buffer,"run a command now")==0)
                                    {
                                        int child;
                                        if ((child = fork())==-1)
                                        {
                                        perror("Could not fork");
                                        }

                                        system("");
                                        printf("Running command\n");
                                        return 0;

                                    }

                                    //read in new job list
                                    if (strcmp(buffer,"load new job list")==0)
                                    {
                                        int njobs=0;
                                        rec=recv(client, &njobs, sizeof(int),
MSG_WAITALL);
                                        printf("I should be getting %i jobs

\n",njobs);
                                        for (i=0;i<njobs;i++)
                                        {
                                                rec=recv(client, &job, sizeof(struct
my_job), MSG_WAITALL);
                                                printf("rec %i : %s %s %s %s %c
\n",rec,job.node,job.path,job.command,job.id,job.status);

add_job(job.id,job.path,job.command,job.node,jobs,job.came_from,job.status,job.pid);
                                        }

                                //printf("%s\n %s\n\n",list[0].id,list[1].path);
                                    debug_print_list(jobs);
                                    }



                                    if (strcmp(buffer,"get number of jobs")==0)
                                    {
                                    printf("I have been asked for all my jobs\n");
                                send_jobs=get_number_of_jobs(jobs);
                                    send(client, &send_jobs,sizeof(int), 0);
                                    printf("The number of jobs is %i",send_jobs);
                                for (i=0;i<max_jobs;i++)
                                    {
                                        if (jobs[i].status!='e')
                            //the job must have a name ie be there
                                        {
                                                //printf("loop %s\n",jobs[i].id);
```

```
                                                    memcpy(&job,&jobs[i],sizeof(struct
my_job));
                                                    strcpy(job.came_from,my_name);
                                                    send(client, &job,sizeof(struct
my_job), 0);
                                                    printf("sending: %s %s %s %s %c
\n",job.node,job.path,job.command,job.id,job.status) ;
                                                }
                                        }

                                        }


                                        if (strcmp(buffer,"close connection")==0)
                                        {
                                            printf("closing conection\n");
                                            //get_times();
                                            close_con=true;
                                        }
                                    printf("%s \n \n",buffer);
                                    //send(clientfd,buffer, rec, 0);
                                    }else
                                    {
                                    printf("size of rec %i\n",rec);
                                    }
                                }
                                close(client);
                                printf("Conection closed\n");


                                close_con=false;

                    }
                    return 0;
}
```

7.3.2 Linux cluster - client.c

```
#include <stdarg.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/socket.h>
#include <resolv.h>
#include <errno.h>
#include "vars.h"
#include <unistd.h>

#define MAXBUF  1024
void PANIC(char *msg);
#define PANIC(msg)  {perror(msg); abort();}

#define                   true         1
#define                   false        0
#define                   max_jobs 100
char suposed_computers[255][20] =
{"134.169.30.83","134.169.30.35","134.169.30.86","134.169.30.94","134.169.30.138","134.16
9.30.0","134.169.30.177","134.169.30.92","134.169.30.85"};
int suposed_port_number[255]= {1250,1250,1250,1250,1250,1250,1250,1250,1250};
int n_sup_computers=10;

int find_least_loaded_computer_from_list(float* computer_load,int ncomputers);


void clear_all_jobs(struct my_job* list,int max)
{
```

```
int i=0;
    for (i=0;i<max_jobs;i++)
    {
             strcpy(list[i].id,"");
             list[i].status='e';
    }
}


int get_number_of_jobs(struct my_job* list)
{
int i=0;
int found=0;
    for (i=0;i<max_jobs;i++)
    {
             if (strcmp(list[i].id,"")!=0)                                  //the job
must have a name ie be there
             {
             found++;
             }
    }

return found;
}


int get_job_id(struct my_job* list,char *id)
{
int i=0;
int found =-1;
    for (i=0;i<max_jobs;i++)
    {
             if (strcmp(list[i].id,id)==0)                                  //the job
must have a name ie be there
             {
             found=i;
             }
    }

return found;
}


//this is an update command and an add command if the ID is allredy there then the item
is updated
// New ver for the client
void add_job_cli(char* id,char* path,char* command,char* node,struct my_job* list,char
status,char *came_from,int pid)
{
int i=0;
int pos=get_job_id(list,id);
    if (pos==-1)                      //if it is not found find a gap and add the job in
    {
             for (i=0;i<max_jobs;i++)
             {
                 if (strcmp(list[i].id,"")==0)                  //ie the list is empty
                 {
                     strcpy(list[i].id,id);
                     strcpy(list[i].path,path);
                     strcpy(list[i].command,command);
                     strcpy(list[i].node,node);
                          strcpy(list[i].came_from,came_from);
                          list[i].status=status;
                          list[i].pid=pid;
                     break;                                                 //break the
for loop early
                 }
             }
    }else                 //if we have found a job pos just update the job info struct
    { //but it is silley if every has the same proority to update
             //if (strcmp(came_from,list[pos].node)==0)         //only update if the
data is comming from the corect computer
```

```
            //if (list[pos].status!='f')                        //if one computer sais
that it is finished then dont let any other
            //if (!((status=='r')&&(strcmp(list[pos].node,came_from)!=0)))
            //only alow a computer which is working on the work to update the state
            //if (!((status=='n')&&(strcmp(list[pos].node,came_from)!=0)))
            //only alow a computer which is working on the work to update the state
            if (strcmp(list[pos].node,came_from)==0)                          //any body
can add the first job, but only the person who it belongs to can update it.
            {
                    strcpy(list[pos].id,id);
                    strcpy(list[pos].path,path);
                    strcpy(list[pos].command,command);
                    strcpy(list[pos].node,node);
                         strcpy(list[pos].came_from,came_from);
                         list[pos].pid=pid;
                         list[pos].status=status;
            }
    }
}

void remove_job(struct my_job* list,char* id)
{
int i=0;
    for (i=0;i<max_jobs;i++)
    {
            if (strcmp(list[i].id,id)==0)                     //ie the list is empty
            {
                list[i].status='e';
                break;                                        //break the for loop
early
            }
    }

}

void debug_print_list(struct my_job* list)
{
int i=0;
    for (i=0;i<max_jobs;i++)
    {
            if (strcmp(list[i].id,"")!=0)                     //ie the list is empty
            {
                printf("QPOS: '%s' '%s' '%s' '%s' %c PID:
%i\n",list[i].node,list[i].path,list[i].command,list[i].id,list[i].status ,list[i].pid) ;
            }
    }
}


int is_the_computer_on(char* computer,int port)
{
            struct sockaddr_in addr;
            char buffer[100];
//          struct sigaction act;

int         serverfd = socket(PF_INET, SOCK_STREAM, 0);
            bzero(&addr, sizeof(addr));
            addr.sin_family = AF_INET;
            addr.sin_port = htons(port);
            inet_aton(computer, &addr.sin_addr);

            if ( connect(serverfd, (struct sockaddr*)&addr, sizeof(addr)) == 0 )
            {
                strcpy(buffer,"would you like a name");
                send(serverfd, buffer,100, 0);
                strcpy(buffer,computer);
                send(serverfd, buffer,100, 0);
                close(serverfd);
                return true;
            }else
            {
                //close(serverfd);
```

```
                return false;
            }

}

int get_q_jobs(struct my_job* list,char* computer,int port)
{
            char buffer[100];
            struct sockaddr_in addr;
            int          bytes_read;
            int cli_jobs=0;
            int i=0;
            struct my_job job;
            int          serverfd = socket(PF_INET, SOCK_STREAM, 0);
            bzero(&addr, sizeof(addr));
            addr.sin_family = AF_INET;
            addr.sin_port = htons(port);
            inet_aton(computer, &addr.sin_addr);

            if ( connect(serverfd, (struct sockaddr*)&addr, sizeof(addr)) == 0 )
            {

            strcpy(buffer,"get number of jobs");
            send(serverfd, buffer,100, 0);

        bzero(buffer, sizeof(buffer));
        bytes_read = recv(serverfd, &cli_jobs, sizeof(int), 0);
             //printf("I should be getting %i jobs \n",cli_jobs);
        for (i=0;i<cli_jobs;i++)
             {
            bytes_read = recv(serverfd, &job, sizeof(struct my_job), MSG_WAITALL);

add_job_cli(job.id,job.path,job.command,job.node,list,job.status,computer,job.pid);
                 //printf("recv %i: from %s %s %s %s %s %c
\n",bytes_read,computer,list[i].node,list[i].path,list[i].command,list[i].id,list[i].stat
us) ;

//void add_job_cli(char* id,char* path,char* command,char* node,struct my_job* list,char
status,char *came_from)
            }


            //printf("%s\n %s\n\n",list[0].id,list[1].path);

            strcpy(buffer,"close connection");
            send(serverfd, buffer,100, 0);

            close(serverfd);

            }
return true;
}



int update_cluster_list(char** computers,int* ports)
{
//lets do this the norty way and scan the sub net.
int i=0;
int ncomputers=0;
char buffer[20];
for (i=0;i<n_sup_computers;i++)
{
    if (is_the_computer_on(suposed_computers[i],suposed_port_number[i])==true)
    {
            strcpy(computers[ncomputers],suposed_computers[i]); //place our found
computer in the list
            ports[ncomputers]=suposed_port_number[i];
            ncomputers++;
//          printf("%s is ours\n",suposed_computers[i]);
    }
}
```

```
return ncomputers;
}

void update_work_list(char** computers,int *port,int ncomputers,struct my_job *jobs)
{
//lets do this the norty way and scan the sub net.
int i=0;

char buffer[20];
for (i=0;i<ncomputers;i++)
{
//printf("going through %s @ %i\n",computers[i],port[i]);
    get_q_jobs(jobs,computers[i],port[i]);
}


/*for (i=0;i<255;i++)                  //port scan
{
    sprintf(buffer,"%s%i",sub_net,i);
    printf("Looking at %s\n",buffer);
    if (is_the_computer_on(buffer)==true)
    {
             strcpy(computers[i],buffer);           //place our found computer in the
list
             ncomputers++;
             printf("%s is ours\n",buffer);
    }
}*/
}

/*int find_computer_pos_in_index(char* computer_name,char** computers,int ncomputers)
{
int n=0;
    for (n=0;n<ncomputers;n++)
    {
             //printf("going through %s @ %i\n",computers[i],port[i]);
             //get_q_jobs(jobs,computers[i],port[i]);
             if (strcmp(computers[n],computer_name)==0)
             {
                 return n;
             }
    }
return -1;
}*/

void compile_computer_load_list(float* computer_load,char** computers,int
ncomputers,struct my_job* jobs)
{
int i=0;
int ii=0;
int res_pos;
    for (i=0;i<ncomputers;i++)
    {
    //printf("going through %s @ %i\n",computers[i],port[i]);
             for(ii=0;ii<max_jobs;ii++)
             {
                 if (jobs[ii].status!='e')
                 if (strcmp(jobs[ii].node,computers[i])==0)
                 if (jobs[ii].status!='f')
                 {
                          computer_load[i]++;
                 }
             }
    }

}



int find_least_loaded_computer(float* computer_load,int ncomputers)
```

```
{
//lets do this the norty way and scan the sub net.
int i=0;
int last_comp=0;
float last_min=computer_load[0];
for (i=0;i<ncomputers;i++)
{
    if (last_min>computer_load[i])
    {
        last_min=computer_load[i];
             last_comp=i;
    }
//    printf("going through %s @ %i\n",computers[i],port[i]);
}

return last_comp;
}

void print_computers_load(float* computer_load,char** computers,int ncomputers)
{
int i=0;
    for (i=0;i<ncomputers;i++)
    {
        printf("%s : %f\n",computers[i],computer_load[i]);
    }

}

void remove_done_jobs(struct my_job* jobs)
{
int i=0;
    for (i=0;i<max_jobs;i++)
    {
             //if (jobs[i].status!='e')                          //ie the list is empty
             if (jobs[i].status=='f')
             {
//               printf("QPOS: %i %s %s %s %s %c
\n",i,list[i].node,list[i].path,list[i].command,list[i].id,list[i].status) ;
                  jobs[i].status='e';                //erase the job
             }
    }

}

int find_next_index_value(struct my_job* jobs)
{
int job_id=-1;
int i=0;
int found=true;
char temp[20];
    do
    {
    job_id++;

             found=false;
             sprintf(temp,"%i",job_id);
             for(i=0;i<max_jobs;i++)
             {
                 if (strcmp(temp,jobs[i].id)==0)
                 {
                 found=true;
                 }
             }

//           if (found==false)
//           {
//           printf("Have found\n");
//
//           }
    }while(found==true);

return job_id;
```

```
}

void send_job_list_to_a_computer(char* computer,int port,struct my_job* jobs)
{

             struct sockaddr_in addr;
             char buffer[100];
             int i;
//           struct sigaction act;
             int send_jobs;
             int sent;
             int          serverfd = socket(PF_INET, SOCK_STREAM, 0);
             bzero(&addr, sizeof(addr));
             addr.sin_family = AF_INET;
             addr.sin_port = htons(port);
             inet_aton(computer, &addr.sin_addr);

             if ( connect(serverfd, (struct sockaddr*)&addr, sizeof(addr)) == 0 )
             {
                 //tell the computer we desire to send a new job list
                 strcpy(buffer,"load new job list");
                 send(serverfd, buffer,100, 0);

                 send_jobs=get_number_of_jobs(jobs);
                 send(serverfd, &send_jobs,sizeof(int), 0);
                 //printf("The number of jobs is %i\n",send_jobs);
                 for (i=0;i<max_jobs;i++)
                 {
                          if (strcmp(jobs[i].id,"")!=0)                      //the job
must be there
                          {
                                       sent=send(serverfd, &jobs[i],sizeof(struct
my_job), 0);
                                       //printf("sending %i: from %s %s %s %s %s %c
\n",sent,computer,jobs[i].node,jobs[i].path,jobs[i].command,jobs[i].id,jobs[i].status) ;

                          }
                 }


                          close(serverfd);
             }

}

void update_computers_job_list(char** computers,int* ports,int ncomputers,struct my_job*
jobs)
{
int i=0;
    for (i=0;i<ncomputers;i++)
    {
             //printf("updateing %s\n",computers[i]);
        send_job_list_to_a_computer(computers[i],ports[i],jobs);
    }

}

//If a job is running on a non existant computer, the job is recoveded and restored to
the waiting condition
int recover_lost_jobs(char** computers,float* computer_load,int ncomputers,struct my_job*
jobs)
{
int computer_down=false;
int i=0;
int ii=0;
int found=false;
int ll_comp=0;                           //least loaded computer
for (i=0;i<max_jobs;i++)
{
found=false;
    for (ii=0;ii<ncomputers;ii++)
    {
```

```
            if (strcmp(jobs[i].node,computers[ii])==0)
            {
            found=true;
            break;
            }
    }

    if (found==false)
    {
    //the command is missing a computer on which to run.
    jobs[i].status='w';                //put the job in to wait mode
    //ll_comp=find_least_loaded_computer(computer_load,ncomputers);
    //computer_load[ll_comp]++;
    //strcpy(jobs[i].node,computers[ll_comp]);
    computer_down=true;
    }
}
return computer_down;
}

//should evnley distrubute the jobs through the computers again
void spread_out_jobs(char** computers,float* computer_load,int ncomputers,struct my_job*
jobs)
{
int i=0;
int ii=0;
int found=false;

for (i=0;i<max_jobs;i++)
{
found=false;
    if (jobs[i].status=='w')                                    //if the job is waiting
then it can be spread out
    {

//          compile_computer_load_list(computer_load,computers,ncomputers,jobs);
                        //re do the calculatin of the load bit slow but never mind
//          ll_comp=find_least_loaded_computer(computer_load,ncomputers);
//          computer_load[ll_comp]++;

strcpy(jobs[i].node,computers[find_least_loaded_computer_from_list(computer_load,ncompute
rs)]);                    //re asign work


    }


}

}

int find_least_loaded_computer_from_list(float* computer_load,int ncomputers)
{
int i=0;
int pos_min_load=0;
int found=false;
int cur_min_load=100000;              //least loaded computer
for (i=0;i<ncomputers;i++)
{
//found=false;
//    if (jobs[i].status=='w')                                    //if the job is waiting
then it can be spread out
//    {

//          compile_computer_load_list(computer_load,computers,ncomputers,jobs);
                        //re do the calculatin of the load bit slow but never mind
//          ll_comp=find_least_loaded_computer(computer_load,ncomputers);
//          computer_load[ll_comp]++;
//          strcpy(jobs[i].node,computers[cur_comp]);                       //re asign
work
//        cur_comp++;
```

```
//           if (cur_comp==ncomputers)
//           {
//               cur_comp=0;
//           }
    if (cur_min_load>computer_load[i])
    {
             cur_min_load=computer_load[i];
             pos_min_load=i;
    }

//    }


}
return pos_min_load;
}


int main(int Count, char *arg_list[])
{
struct my_job jobs[max_jobs];          //list of jobs to be worked on
char path[300];                                    //A path of 300 is a bit small but
never mind
char temp[20];
char **computers;
int *ports;
int ll_node=0;                         //least loaded node
float *computer_load;
    int i;
    int k;
clear_all_jobs(jobs,max_jobs);


    computers = (char**)calloc(255,sizeof(char*));
    for (i=0;i<255;i++)
    {
             computers[i]=(char*)calloc(20,sizeof(char));
    }

    ports = (int*)calloc(255,sizeof(int));
    computer_load = (float*)calloc(255,sizeof(float));
    bzero(computer_load, sizeof(int)*255);


int ncomputers;
if ((ncomputers=update_cluster_list(computers,ports))==0)                    //find
which computers in the cluster are switched and if they are switched on update there name
{
printf ("No computers were found sory!");
return 0;
}
printf ("I have found computers in the cluster %i\n ",ncomputers);

//get_q_jobs(jobs,"134.169.30.92");
update_work_list(computers,ports,ncomputers,jobs);              //this line should
compile a work list from all the computers hoho

//print the jobs found from every client
//debug_print_list(jobs);
//this should extract from the job list how many jobs each computer has
compile_computer_load_list(computer_load,computers,ncomputers,jobs);
print_computers_load(computer_load,computers,ncomputers);
ll_node=find_least_loaded_computer(computer_load,ncomputers);
//printf("The least loaded node is %s\n",computers[ll_node]);

//gets the next job id
int next_id;
//next_id=find_next_index_value(jobs);
//printf("The next ID is %i\n",next_id);
//sprintf(temp,"%i",next_id);

//set up the job to be sent
```

```
//    struct my_job job;
//    job.status='w';
//    strcpy(job.id,temp);
//    strcpy(job.node,computers[ll_node]);
//    strcpy(job.path," /root/");
//    strcpy(job.command,"find ./");
//now add this newley  built job to the list

//add our job to the job list
//debug_print_list(jobs);
   getcwd(path, 300);
//printf("I have found the path as %s",  path);
    if (strcmp(arg_list[1],"--kill")==0)
    {
             //add_job_cli(arg_list[2],path,"",computers[ll_node],jobs,'e',"Server");
             remove_job(jobs,arg_list[2]);
    }


    if (strcmp(arg_list[1],"--add")==0)
    {
    next_id=find_next_index_value(jobs);
    //printf("The next ID is %i\n",next_id);
    sprintf(temp,"%i",next_id);

    add_job_cli(temp,path,arg_list[2],computers[ll_node],jobs,'w',"Server",0);
    }

    if (strcmp(arg_list[1],"--addall")==0)
    {
        for (k=0;k<ncomputers;k++)
             {
//             printf("adding job to all computers %s\n",computers[k]);
                 next_id=find_next_index_value(jobs);
                 //printf("The next ID is %i\n",next_id);
                 sprintf(temp,"%i",next_id);

                 add_job_cli(temp,path,arg_list[2],computers[k],jobs,'w',"Server",0);
             }
    }


    if (strcmp(arg_list[1],"--view")==0)
    {
    debug_print_list(jobs);
    //add_job_cli(temp,path,arg_list[2],computers[ll_node],jobs,'w',"Server");
    }

//    debug_print_list(jobs);
    //if (strcmp(arg_list[1],"--redisp")==0)
    //{
    //(jobs);
    //add_job_cli(temp,path,arg_list[2],computers[ll_node],jobs,'w',"Server");
    //}


//    printf("before\n");
//    add_job_cli(temp,path,"find ./",computers[ll_node],jobs,'w',"Server");
    remove_done_jobs(jobs);
//    debug_print_list(jobs);

//    send_job_list_to_a_computer(computers[ll_node],ports[ll_node],jobs);
    if (strcmp(arg_list[1],"--spread")==0)
    {

             if (recover_lost_jobs(computers,computer_load,ncomputers,jobs)==true)
             {
                 spread_out_jobs(computers,computer_load,ncomputers,jobs);
             }

    debug_print_list(jobs);
    }
```

```
    //debug_print_list(jobs);

    update_computers_job_list(computers,ports,ncomputers,jobs);



    //debug_print_list(jobs);

    free(ports);
    free(computer_load);

    for (i=0;i<255;i++)
    {
    free(computers[i]);
    }
    free(computers);


return 0;
}
```