CS 180A Assignment 3 (Matrix) - Ryman Barnett

Generated by Doxygen 1.8.17

1 File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

2 File Documentation

2.1 child-matrix.c File Reference

Source file for child-matrix functions to calculate the value of a single position in a matrix squared.

```
#include <stdlib.h>
#include <sys/shm.h>
#include <stdio.h>
```

Functions

- int main (int argc, char **argv)

  calculates the value of a single position in a matrix squared

2.1.1 Detailed Description

Source file for child-matrix functions to calculate the value of a single position in a matrix squared.

Author

Ryman Barnett

email: ryman.b@digipen.edu

DigiPen login: ryman.b

Course: CS180

Section: A

Assignment #3

Date

2022-10-14

2.1.2 Function Documentation

2.1.2.1 main()

```
int main (
        int argc,
        char ** argv )
```

calculates the value of a single position in a matrix squared

Parameters

argc	number of arguments
argv	array of arguments

Returns

0 on success, error code on failure

Definition at line 32 of file child-matrix.c.

```
33 {
34
35   int childNum;     /* what child am I? */
36   int row;          /* which row am I? */
37   int col;          /* which column am I? */
38   int shmid;        /* shared memory id */
39   int* buffer;      /* buffer for shared mem */
40   int i;            /* loop index            */
41   int width;        /* width of the matrix   */
42   int sum = 0;      /* sum of the row        */
43
44   /* Check for the correct number of parameters */
45   if (argc < 5)
46   {
47     printf("Insufficient parameters supplied %i\n", argc);
48     exit(-1); /* exit with error -1 */
49   }
50
51   /* get the shared memory id */
52   shmid = atoi(argv[1]);
53   /* get the child number */
54   childNum = atoi(argv[2]);
55   /* get the row of the matrix to use*/
56   row = atoi(argv[3]);
57   /* get the column of the matrix to use*/
58   col = atoi(argv[4]);
59
60   /* attach shared memory */
61   buffer = (int *) shmat(shmid, NULL, 0);
62   /* if failed exit */
63   if (buffer == (int*)-1)
64   {
65     perror("shmat");
66     exit (-2); /* exit with error -2 */
67   }
68
69   /* get the width of the matrix */
70   width = *buffer;
71   /* calculate the sum of the row */
72   for (i = 0; i < width; i++)
73   {
74     sum += buffer[1 + (row * width) + i] * buffer[1 + col + (i * width)];
75   }
76
77   buffer[childNum] = sum; /* set position in shared mem to answer */
78
79   /* cleanup memory*/
80   shmdt(buffer);  /* detach memory from child process */
81
82   exit(0); /* exit with no error */
83 }
```

2.2 parent-matrix.c File Reference

Source file for parent-matrix functions to create a matrix from a file, print a matrix, and fork a matrix. into child processes that calculate matrix squared.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <unistd.h>
#include <sys/wait.h>
```

Macros

- #define FILENAME_MAX 4096

Functions

- int * get_matrix (const char *filename, int *width)

  creates a matrix from a file
- void print_matrix (int *matrix, int width)

  prints a matrix
- int main (int argc, char **argv)

  handles creating a matrix, setting up shared memory, and forking a child process for each row of the matrix to calculate the square of the matrix

2.2.1 Detailed Description

Source file for parent-matrix functions to create a matrix from a file, print a matrix, and fork a matrix. into child processes that calculate matrix squared.

Author

Ryman Barnett

email: ryman.b@digipen.edu

DigiPen login: ryman.b

Course: CS180

Section: A

Assignment #3

Date

2022-10-14

2.2.2 Macro Definition Documentation

2.2.2.1 FILENAME_MAX `#define FILENAME_MAX 4096`

max filename length

Definition at line 23 of file parent-matrix.c.

2.2.3 Function Documentation

2.2.3.1 get_matrix() `int* get_matrix ( const char * filename, int * width )`

creates a matrix from a file

Parameters

filename	name of file to read from
width	width of the matrix

Returns

pointer to the matrix

Definition at line 38 of file parent-matrix.c.

```
39 {
40   int value, *matrix; /* array for matrix */
41   FILE *fp;           /* file to open */
42
43   /* Open the file in text/translated mode */
44   fp = fopen(filename, "rt");
45   /* Check for error */
46   if (!fp)
47   {
48     printf("Can't open file: %s\n", filename);
49     exit(-1); /* exit with error -1 */
50   }
51
52   /* Read the width and allocate the matrix */
53   fscanf(fp, "%d", width);
54   matrix = (int*)malloc(*width * *width * sizeof(int));
55   if (!matrix)
56   {
57     printf("Can't malloc matrix\n");
58     fclose(fp); /* close the file */
59     exit (-2);  /* exit with error -2 */
60   }
61
62   /* Read the vaules and put in the matrix */
63   while (!feof(fp))
```

```
64   {
65     int result = fscanf(fp, "%d", &value);
66     if (result == -1)
67       break;
68     *matrix++ = value;
69   }
70   fclose(fp); /* close the file */
71
72   /* Return the address of the matrix */
73   return matrix - (*width * *width);
74 }
```

Referenced by main().

2.2.3.2 main() int main (int *argc*, char ** *argv*)

handles creating a matrix, setting up shared memory, and forking a child process for each row of the matrix to calculate the square of the matrix

Parameters

argc	number of arguments
argv	array of arguments

Returns

0 on success, error code on failure

Definition at line 113 of file parent-matrix.c.

```
114 {
115   int width;       /* width of the matrix   */
116   int *matrix;     /* the matrix read in    */
117   int *cPid;       /* child process id      */
118   key_t key = 123; /* arbitrary key (0x7b)  */
119   int shmid;       /* return value from fork/shmget */
120   int* buffer;     /* buffer for shared mem */
121   int i;           /* loop index            */
122   int res;         /* return value from execv */
123   size_t size;     /* size of shared memory */
124
125   /* Check for the correct number of parameters */
126   if (argc < 3)
127   {
128     printf("Insufficient parameters supplied\n");
129     return -1;
130   }
131
132   /* read in matrix values from file */
133   matrix = get_matrix(argv[1], &width);
134
135   /* print the matrix */
136   print_matrix(matrix, width);
137
138   /* allocate shared memory */
139   size = (sizeof(int) * (width * width) * 2) + 1;
140   shmid = shmget(key, size, 0600 | IPC_CREAT);
141   /* if failed exit */
142   if (shmid == -1)
143   {
144     perror("shmget");
```

```
145     free (matrix); /* free the matrix */
146     exit(1);       /* exit with error 1*/
147   }
148
149   /* attach shared memory */
150   buffer = (int *) shmat(shmid, NULL, 0);
151   /* if failed exit */
152   if (buffer == (int*)-1)
153   {
154     perror("shmat");
155     free (matrix);                      /* free the matrix */
156     if(shmctl(shmid, IPC_RMID, 0) == -1) /* free memory block */
157     {
158       perror("shmctl");
159     }
160     exit(2);                            /* exit with error 2 */
161   }
162
163   /* copy the width and matrix into shared memory */
164   buffer[0] = width;
165   for (i = 1; i < (width * width) + 1; i++)
166   {
167     buffer[i] = matrix[i - 1];
168   }
169
170   free(matrix);               /* free the matrix                  */
171
172   /* create child process list */
173   cPid = (int*)malloc(sizeof(int) * (width * width));
174
175   /* Fork a child for each matrix entry and put into child array*/
176   for (i = 0; i < width * width; i++)
177   {
178     /* fork a child */
179     cPid[i] = fork();
180
181     /* if child, exec the child program */
182     if (cPid[i] == 0)
183     {
184       char key[FILENAME_MAX]; /* shmid */
185       char pos[FILENAME_MAX]; /* position in matrix of child*/
186       char row[FILENAME_MAX]; /* row of child */
187       char col[FILENAME_MAX]; /* column of child */
188       char *args[6] = {"\0"}; /* arguments for child(Null terminated) */
189
190       /*convert ints to strings for execution */
191       if(sprintf(key, "%d", shmid) < 0)
192       {
193         perror("sprintf");
194         exit(-1); /* exit with error -1 */
195       }
196       if(sprintf(pos, "%d", (i + 1) + width * width) < 0)
197       {
198         perror("sprintf");
199         exit(-1); /* exit with error -1 */
200       }
201       if(sprintf(row, "%d", i / width) < 0)
202       {
203         perror("sprintf");
204         exit(-1); /* exit with error -1 */
205       }
206       if(sprintf(col, "%d", i % width) < 0)
207       {
208         perror("sprintf");
209         exit(-1); /* exit with error -1 */
210       }
211
212       /* put arguments into array */
213       args[0] = argv[2];
214       args[1] = key;
215       args[2] = pos;
216       args[3] = row;
217       args[4] = col;
218
219       free(cPid); /* free the child pid list in the children*/
220
221       /* execute child */
222       res = execv(argv[2], args);
223       /* if execv fails */
224       if (res == -1)
225       {
```

```
226         printf("execv failed with %d", res);
227         exit(-1); /* exit child fail */
228       }
229
230       exit(0); /* exit child */
231     }
232   }
233
234   /* wait for children */
235   for (i = 0; i < width * width; i++)
236   {
237     int status; /* status of child */
238
239     /* wait for all children */
240     if(waitpid(cPid[i], &status, 0) == -1)
241     {
242       perror("waitpid");
243     }
244   }
245
246   /* print matrix from shared buffer */
247   print_matrix(buffer + 1 + width * width, width);
248
249   /* cleanup memory */
250   /* detach memory from parent process */
251   if(shmdt(buffer) == -1)
252   {
253     perror("shmdt");
254   }
255   /* delete memory block */
256   if(shmctl(shmid, IPC_RMID, 0) == -1)
257   {
258     perror("shmctl");
259   }
260   free(cPid); /* free child pid array */
261
262   return 0; /* parent completed success*/
263 }
```

References FILENAME_MAX, get_matrix(), and print_matrix().

2.2.3.3 print_matrix() `void print_matrix (`

```
int * matrix,
int width )
```

prints a matrix

Parameters

matrix	pointer to the matrix
width	width of the matrix

Definition at line 86 of file parent-matrix.c.

```
87 {
88   int i, size = width * width;
89   for (i = 0; i < size; i++)
90   {
91     printf("%8i", matrix[i]);
92     if ( (i + 1) % width == 0)
93       printf("\n");
94   }
95   printf("\n");
96 }
```

Referenced by main().

Index