AT

Generated by Doxygen 1.8.13

Contents

Chapter 1

At

Members

- 15/0032552 | Claudio Segala Rodrigues Silva Filho
- 00/0000000 | [Cristiano Cardoso]()
- 15/0137885 | Luís Eduardo Luz Silva
- 14/0033599 | [Yan Victor]()

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 job Struct Reference

Um job com toda a estrutura que compõe o job.

`#include <job.h>`

Data Fields

- int id
- int seconds
- int delay
- char filename [1000]
- bool done
- int submission
- int start
- int finish
- int completed
- int node_time [N]
- int node_pid [N]

4.1.1 Detailed Description

Um job com toda a estrutura que compõe o job.

Definition at line 20 of file job.h.

4.1.2 Field Documentation

4.1.2.1 completed

```
int completed
```

Definition at line 29 of file job.h.

4.1.2.2 delay

```
int delay
```

Definition at line 23 of file job.h.

4.1.2.3 done

```
bool done
```

Definition at line 25 of file job.h.

4.1.2.4 filename

```
char filename[1000]
```

Definition at line 24 of file job.h.

4.1.2.5 finish

```
int finish
```

Definition at line 28 of file job.h.

4.1.2.6 id

```
int id
```

Definition at line 21 of file job.h.

4.1.2.7 node_pid

```
int node_pid[N]
```

Definition at line 31 of file job.h.

4.1.2.8 node_time

```
int node_time[N]
```

Definition at line 30 of file job.h.

4.1.2.9 seconds

```
int seconds
```

Definition at line 22 of file job.h.

4.1.2.10 start

```
int start
```

Definition at line 27 of file job.h.

4.1.2.11 submission

```
int submission
```

Definition at line 26 of file job.h.

The documentation for this struct was generated from the following file:

- server/job.h

4.2 list Struct Reference

A list. A doubly linked list.

```
#include <list.h>
```

Data Fields

- int sz

  The size of the list
- struct node * begin

  Pointer the beginning of the list
- struct node * end

  Pointer to the beginning of the list

4.2.1 Detailed Description

A list. A doubly linked list.

Definition at line 35 of file list.h.

4.2.2 Field Documentation

4.2.2.1 begin

```
struct node* begin
```

Pointer the beginning of the list

Definition at line 40 of file list.h.

4.2.2.2 end

```
struct node* end
```

Pointer to the beginning of the list

Definition at line 43 of file list.h.

4.2.2.3 sz

```
int sz
```

The size of the list

Definition at line 37 of file list.h.

The documentation for this struct was generated from the following file:

- server/list.h

4.3 msgbuf Struct Reference

Message The structure of the message that will be sent between client and server.

`#include <message.h>`

Data Fields

- long type
  The type of the message
- long id
  The job identifier
- int t
  The time in seconds since 70s
- int delay
  The delay in seconds
- int origin
- char s [MAX_STRING_SIZE]
  The path to the executables

4.3.1 Detailed Description

Message The structure of the message that will be sent between client and server.

Message.

The structure of the message that will be sent between client and server

Definition at line 9 of file message.h.

4.3.2 Field Documentation

4.3.2.1 delay

`int delay`

The delay in seconds

Definition at line 20 of file message.h.

4.3.2.2 id

```
long id
```

The job identifier

Definition at line 14 of file message.h.

4.3.2.3 origin

```
int origin
```

Definition at line 23 of file message.h.

4.3.2.4 s

```
char s
```

The path to the executables

Definition at line 26 of file message.h.

4.3.2.5 t

```
int t
```

The time in seconds since 70s

Definition at line 17 of file message.h.

4.3.2.6 type

```
long type
```

The type of the message

Definition at line 11 of file message.h.

The documentation for this struct was generated from the following file:

- client/message.h

4.4 node Struct Reference

A node. The node of a doubly linked list.

`#include <list.h>`

Data Fields

- struct node * prev
  Pointer to the node before
- struct node * nxt
  Pointer to the following node
- void * value
  Void Pointer to the value

4.4.1 Detailed Description

A node. The node of a doubly linked list.

Definition at line 20 of file list.h.

4.4.2 Field Documentation

4.4.2.1 nxt

`struct node* nxt`

Pointer to the following node

Definition at line 25 of file list.h.

4.4.2.2 prev

`struct node* prev`

Pointer to the node before

Definition at line 22 of file list.h.

4.4.2.3 value

```
void* value
```

Void Pointer to the value

Definition at line 28 of file list.h.

The documentation for this struct was generated from the following file:

- server/list.h

4.5 semaphore Struct Reference

Data Fields

- int id
- struct sembuf def [2]

4.5.1 Detailed Description

Definition at line 3 of file semaphore.c.

4.5.2 Field Documentation

4.5.2.1 def

```
struct sembuf def[2]
```

Definition at line 5 of file semaphore.c.

4.5.2.2 id

```
int id
```

Definition at line 4 of file semaphore.c.

The documentation for this struct was generated from the following file:

- util/semaphore.c

Chapter 5

File Documentation

5.1 client/main.c File Reference

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/msg.h>
#include <sys/stat.h>
#include "message.h"
```

Macros

- #define KEY 15003
- #define N 15
- #define S(x) printf("\033[0;32m%s\033[0m\n", x);
- #define E(x) printf("\033[0;31m%s\033[0m\n", x);

Functions

- bool try_cast_int (char *num, int *result)

  Cast string to int.
- void send (int seconds, char filename[ ])

  Send a message with a job to the queue.
- void destroy ()

  Destroy the queue.
- int main (int argc, char *argv[ ])

5.1.1 Macro Definition Documentation

5.1.1.1 E

```
#define E(
              x ) printf("\033[0;31m%s\033[0m\n", x);
```

Definition at line 18 of file main.c.

5.1.1.2 KEY

```
#define KEY 15003
```

Definition at line 14 of file main.c.

5.1.1.3 N

```
#define N 15
```

Definition at line 15 of file main.c.

5.1.1.4 S

```
#define S(
              x ) printf("\033[0;32m%s\033[0m\n", x);
```

Definition at line 17 of file main.c.

5.1.2 Function Documentation

5.1.2.1 destroy()

```
void destroy ( )
```

Destroy the queue.

Definition at line 74 of file main.c.

```
74                {
75     int id = msgget(KEY, 0);
76
77     printf("Destroying the queue %d\n", id);
78     struct msqid_ds msg;
79     msgctl(id, IPC_RMID, &msg);
80 }
```

5.1.2.2 main()

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

Definition at line 82 of file main.c.

```
82                                  {
83     int seconds = 0;
84
85     if (argc == 1) {
86         destroy();
87     }
88
89     if (argc == 3 && try_cast_int(argv[1], &seconds) && argv[2] != NULL) {
90         char filename[MAX_STRING_SIZE];
91
92         strcpy(filename, argv[2]);
93
94         send(seconds, filename);
95     }
96
97     return 0;
98 }
```

5.1.2.3 send()

```
void send (
              int seconds,
              char filename[] )
```

Send a message with a job to the queue.

Parameters

t	the time in which the job should be executed.
filename	the path of the file to be executed.

Definition at line 45 of file main.c.

```
45                                          {
46     int key = KEY; // matricula truncada
47     int id = msgget(key, S_IWUSR);
48
49     time_t t = time(NULL) + seconds;
50
51     if (id < 0) {
52         E("Failed to get queue");
53     } else {
54         S("Got the queue");
55     }
56
57     srand(time(NULL) + getpid() + clock());
58
59     Msg msg = { N+1, rand(), t, seconds, N+1 };
60
61     strcpy(msg.s, filename);
62     int res = msgsnd(id, &msg, sizeof(Msg) - sizeof(long), 0);
63
64     if (res < 0) {
65         E("Failed to send messages");
```

```
66     } else {
67         printf("\n> job=%ld, arquivo=%s, delay=%ds\n", msg.id, msg.s, msg.
       delay);
68     }
69 }
```

5.1.2.4 try_cast_int()

```
bool try_cast_int (
            char * num,
            int * result )
```

Cast string to int.

Parameters

num	a string with a number.
result	a pointer to the result of the convertion.

Returns

if it succeeded or not.

Definition at line 26 of file main.c.

```
26                                          {
27     int sz = strlen(num);
28
29     for (int i = sz - 1, n = 1; i >= 0; i--, n *= 10) {
30         if (num[i] >= '0' && num[i] <= '9') {
31             (*result) += n * (num[i] - '0');
32         } else {
33             return false;
34         }
35     }
36
37     return true;
38 }
```

5.2 prog/main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

Functions

- int main ()

5.2.1 Function Documentation

5.2.1.1 main()

```
int main ( )
```

Definition at line 5 of file main.c.

```
5             {
6     sleep(2);
7     printf(".");
8     return 0;
9 }
```

5.3 server/main.c File Reference

```
#include "util.h"
#include "job.h"
#include "list.h"
#include "message.h"
#include "tree.h"
#include "hypercube.h"
#include "torus.h"
```

Macros

- #define SCHEDULER 200
- #define SHUTDOWN 201

Functions

- void destroy ()

  Destroy the list of jobs and the queue.
- Msg mng_execute (int idx, char *program)

  Execute a program.
- void mng_broadcast_down (int idx, Msg msg)

  Send message down the structure.
- void mng_broadcast_up (int idx, Msg msg)

  Send message up the structure.
- void mng_shutdown ()
- void mng_start (int idx)

  Start a manager.
- void mng_create (int n)

  Creation of all management processes.
- void sch_shutdown ()
- Job * sch_get_next_job ()

Retrieves the oldest job that wasn't executed.

- void sch_execute ()

  Send message to node 0 to start execution.

- void sch_msg_error ()

  Try to treat error that might be caused by alarm.

- void sch_msg_success (Msg msg)

  Treat a successfull message coming.

- void sch_try_execute ()

  Verify if the topology is executing a job and then get the next job to execute.

- void sch_start ()

  Start a scheduler.

- int main (int argc, char ∗argv[ ])

  Setup everything, create the managers and start the scheduler.

Variables

- int topology_type
- int structure [N]
- int queue_id
- bool topology_free
- List ∗ jobs
- Job ∗ curr_job

5.3.1 Macro Definition Documentation

5.3.1.1 SCHEDULER

```
#define SCHEDULER 200
```

Definition at line 12 of file main.c.

5.3.1.2 SHUTDOWN

```
#define SHUTDOWN 201
```

Definition at line 13 of file main.c.

5.3.2 Function Documentation

5.3.2.1 destroy()

```
void destroy ( )
```

Destroy the list of jobs and the queue.

Definition at line 30 of file main.c.

```
30                {
31     list_destroy(jobs);
32
33     struct msqid_ds msg;
34     msgctl(queue_id, IPC_RMID, &msg);
35
36     exit(0);
37 }
```

5.3.2.2 main()

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

Setup everything, create the managers and start the scheduler.

Parameters

argc	The number of arguments
argv	The array of arguments

Returns

int If there was an error

Definition at line 335 of file main.c.

```
335                                    {
336     queue_id = msgget(KEY, IPC_CREAT | 0777);
337
338     if (queue_id < 0) {
339         E("Failed to get queue");
340     } else {
341         S("Queue set");
342     }
343
344     jobs = list_create();
345     S("List of jobs set");
346
347     if (argc != 2 || !try_cast_int(argv[1], &topology_type)) {
348         E("Not a valid topology");
349         exit(1);
350     }
351
352     switch (topology_type) {
353         case TREE:
354             ft_make(structure);
355             break;
356         case HYPERCUBE:
357             hc_make(structure);
```

```
358             break;
359         case TORUS:
360             tr_make(structure);
361             break;
362         default:
363             E("Wrong topology!");
364             exit(1);
365     }
366
367     S("Topology set");
368
369     mng_create(N);
370     S("Create Managers");
371
372     sch_start();
373
374     return 0;
375 }
```

5.3.2.3 mng_broadcast_down()

```
void mng_broadcast_down (
              int idx,
              Msg msg )
```

Send message down the structure.

Parameters

idx	The current index
msg	The message to be sent

Definition at line 82 of file main.c.

```
82                                          {
83     int arr[4] = { -1, -1, -1, -1 };
84
85     switch (topology_type) {
86         case TREE:
87             ft_down(idx, arr);
88             break;
89         case HYPERCUBE:
90             hc_down(idx, arr);
91             break;
92         case TORUS:
93             tr_down(idx, arr);
94             break;
95     }
96
97     for (int i = 0; i < 4; i++) {
98         if (arr[i] != -1) {
99             msg.type = arr[i] + 1;
100             msgsnd(queue_id, &msg,  sizeof(Msg) - sizeof(long), IPC_NOWAIT);
101         }
102     }
103 }
```

5.3.2.4 mng_broadcast_up()

```
void mng_broadcast_up (
              int idx,
              Msg msg )
```

Send message up the structure.

Parameters

idx	The current index
msg	The message to be sent

Definition at line 111 of file main.c.

```
111                                         {
112     switch (topology_type) {
113         case TREE:
114             msg.type = ft_up(idx);
115             break;
116         case HYPERCUBE:
117             msg.type = hc_up(idx);
118             break;
119         case TORUS:
120             msg.type = tr_up(idx);
121             break;
122     }
123
124     msg.type++;
125     msgsnd(queue_id, &msg, sizeof(Msg) - sizeof(long), IPC_NOWAIT);
126 }
```

5.3.2.5 mng_create()

```
void mng_create (
              int n )
```

Creation of all management processes.

Definition at line 169 of file main.c.

```
169                           {
170     for (int i = 0; i < n; i++) {
171         pid_t pid = fork();
172
173         if (!pid) {
174             mng_start(i);
175         }
176     }
177 }
```

5.3.2.6 mng_execute()

```
Msg mng_execute (
              int idx,
              char * program )
```

Execute a program.

Parameters

idx	The current index
program	The name of the program

Returns

Msg Message with the elapsed time

Definition at line 46 of file main.c.

```
46                                      {
47     time_t start = time(NULL);
48
49     int pid_worker = fork();
50     if (pid_worker == 0) { // child of the fork
51         char path[100] = PATH;
52         strcat(path, program);
53         int err = execl(path, program, (char *) 0);
54
55         if (err < 0) {
56             E("An error ocurred!");
57             exit(1);
58         }
59     }
60
61     int status;
62     wait(&status);
63     time_t elapsed = time(NULL) - start;
64
65     Msg msg;
66
67     msg.type = 0;
68     msg.id = getpid();
69     msg.t = elapsed;
70     msg.delay = 0;
71     msg.origin = idx;
72
73     return msg;
74 }
```

5.3.2.7 mng_shutdown()

```
void mng_shutdown ( )
```

Definition at line 128 of file main.c.

```
128                     {
129     exit(0);
130 }
```

5.3.2.8 mng_start()

```
void mng_start (
            int idx )
```

Start a manager.

Definition at line 135 of file main.c.

```
135                         {
136     signal(SIGUSR1, mng_shutdown);
137 
138     Msg msg;
139 
140     // msg.type = SHUTDOWN;
141     // msg.id = getpid();
142 
143     // msgsnd(queue_id, &msg, sizeof(Msg) - sizeof(long), IPC_NOWAIT);
144 
145     while (true) {
146         int virtual_queue = idx+1;
147         int res = msgrcv(queue_id, &msg, sizeof(Msg) - sizeof(long), virtual_queue, 0);
148 
149         if (res < 0) {
150             E("Failed to receive message. Maybe the queue died?");
151             break;
152         }
153 
154         if (msg.origin == SCHEDULER) {
155             mng_broadcast_down(idx, msg);
156             msg = mng_execute(idx, msg.s);
157             mng_broadcast_up(idx, msg);
158         } else {
159             mng_broadcast_up(idx, msg);
160         }
161     }
162 
163     exit(1);
164 }
```

5.3.2.9 sch_execute()

```
void sch_execute ( )
```

Send message to node 0 to start execution.

Definition at line 222 of file main.c.

```
222                     {
223     printf("[SCHEDULER] Executing the Job %d\n\n", curr_job->id);
224 
225     topology_free = false;
226     curr_job->start = time(NULL);
227 
228     Msg msg;
229 
230     msg.type = 1;
231     msg.id = 0;
232     msg.t = 0;
233     msg.delay = 0;
234     msg.origin = SCHEDULER;
235     strcpy(msg.s, curr_job->filename);
236 
237     msgsnd(queue_id, &msg, sizeof(Msg) - sizeof(long), 0);
238 }
```

5.3.2.10 sch_get_next_job()

```
Job* sch_get_next_job ( )
```

Retrieves the oldest job that wasn't executed.

Returns

Job* a pointer to the next job

Definition at line 202 of file main.c.

```
202                         {
203     Job *job = NULL;
204     Node *curr = jobs->begin;
205 
206     while (curr != NULL) {
207         Job *aux = (Job*) curr->value;
208 
209         if ((job == NULL || job->seconds > aux->seconds) && !aux->
      done) {
210             job = aux;
211         }
212 
213         curr = curr->nxt;
214     }
215 
216     return job;
217 }
```

5.3.2.11 sch_msg_error()

```
void sch_msg_error ( )
```

Try to treat error that might be caused by alarm.

Definition at line 243 of file main.c.

```
243                     {
244     if (errno != ENOMSG) {
245         E("Failed to receive message");
246         exit(1);
247     }
248 }
```

5.3.2.12 sch_msg_success()

```
void sch_msg_success (
             Msg msg )
```

Treat a successfull message coming.

Parameters

msg	The message received

Definition at line 255 of file main.c.

```
255                             {
```

```
256     printf("\n\n> type: %ld, id: %ld, seconds: %d(Since 70's), delay: %ds, message: %s, origin: %d\n\n",
      msg.type, msg.id, msg.t, msg.delay, msg.s, msg.origin);
257
258     if (msg.origin < N) { // message from sons
259         curr_job->node_pid[msg.origin] = msg.id;
260         curr_job->node_time[msg.origin] = msg.t;
261         curr_job->completed++;
262
263         if (curr_job->completed == N) { // finished the job
264             topology_free = true;
265             curr_job->done = true;
266             curr_job->finish = time(NULL);
267
268             printf("\n> job=%d, arquivo=%s, delay=%ds, makespan: %ds\n",
      curr_job->id, curr_job->filename, curr_job->
      delay, curr_job->finish - curr_job->start);
269         }
270     } else { // message from clients
271         Job *job = job_create(msg.id, msg.t, msg.s, msg.
      delay);
272
273         list_push_back(jobs, job);
274     }
275 }
```

5.3.2.13 sch_shutdown()

```
void sch_shutdown ( )
```

Definition at line 179 of file main.c.

```
179                     {
180     Node *node = jobs->begin;
181
182     while (node != NULL) {
183         Job *job = (Job*)node->value;
184
185         if(job->done) {
186             printf("[DONE] job: %d, file: %s, submission: %d\n", job->id, job->
      filename, job->submission);
187         } else{
188             printf("[CANCELLED] job: %d, file: %s\n", job->id, job->filename);
189         }
190
191         node = node->nxt;
192     }
193
194     destroy();
195 }
```

5.3.2.14 sch_start()

```
void sch_start ( )
```

Start a scheduler.

Definition at line 296 of file main.c.

```
296                 {
297     signal(SIGUSR1, sch_shutdown);
298 
299     Msg msg;
300 
301     // msg.type = SHUTDOWN;
302     // msg.id = getpid();
303 
304     // msgsnd(queue_id, &msg, sizeof(Msg) - sizeof(long), IPC_NOWAIT);
305 
306     int cont = 0;
307     int virtual_id = N+1; //> the virtual queue id
308     char traces[1000];
309 
310     topology_free = true;
311 
312     while (true) {
313         Msg msg;
314         int res = msgrcv(queue_id, &msg, sizeof(Msg) - sizeof(long), virtual_id, IPC_NOWAIT);
315 
316         if (res < 0) {
317             sch_msg_error();
318         } else {
319             sch_msg_success(msg);
320         }
321 
322         sch_try_execute();
323     }
324 
325     destroy();
326 }
```

5.3.2.15 sch_try_execute()

```
void sch_try_execute ( )
```

Verify if the topology is executing a job and then get the next job to execute.

Definition at line 281 of file main.c.

```
281                     {
282     if (topology_free) {
283         Job *nxt_job = sch_get_next_job();
284         time_t now = time(NULL);
285 
286         if (nxt_job && nxt_job->seconds <= now) {
287             curr_job = nxt_job;
288             sch_execute();
289         }
290     }
291 }
```

5.3.3 Variable Documentation

5.3.3.1 curr_job

```
Job* curr_job
```

Definition at line 25 of file main.c.

5.3.3.2 jobs

```
List* jobs
```

Definition at line 23 of file main.c.

5.3.3.3 queue_id

```
int queue_id
```

Definition at line 19 of file main.c.

5.3.3.4 structure

```
int structure[N]
```

Definition at line 17 of file main.c.

5.3.3.5 topology_free

```
bool topology_free
```

Definition at line 21 of file main.c.

5.3.3.6 topology_type

```
int topology_type
```

Definition at line 15 of file main.c.

5.4 shutdown/main.c File Reference

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <signal.h>
#include "message.h"
```

Macros

- #define SCHEDULER 200
- #define SHUTDOWN 201
- #define KEY 15003
- #define N 15
- #define S(x) printf("\033[0;32m%s\033[0m\n", x);
- #define E(x) printf("\033[0;31m%s\033[0m\n", x);

Functions

- int main ()

5.4.1 Macro Definition Documentation

5.4.1.1 E

```
#define E(
              x ) printf("\033[0;31m%s\033[0m\n", x);
```

Definition at line 38 of file main.c.

5.4.1.2 KEY

```
#define KEY 15003
```

Definition at line 34 of file main.c.

5.4.1.3 N

```
#define N 15
```

Definition at line 35 of file main.c.

5.4.1.4 S

```
#define S(
              x ) printf("\033[0;32m%s\033[0m\n", x);
```

Definition at line 37 of file main.c.

5.4.1.5 SCHEDULER

```
#define SCHEDULER 200
```

Definition at line 31 of file main.c.

5.4.1.6 SHUTDOWN

```
#define SHUTDOWN 201
```

Definition at line 32 of file main.c.

5.4.2 Function Documentation

5.4.2.1 main()

```
int main ( )
```

Definition at line 40 of file main.c.

```
40              {
41      int key = KEY; // matricula truncada
42      int queue_id = msgget(key, S_IWUSR);
43
44      if (queue_id < 0) {
45          E("Failed to get queue");
46      } else {
47          S("Got the queue");
48      }
49
50      int count = N + 1;
51
52      while (count--) {
53          Msg msg;
54
55          int virtual_queue = SHUTDOWN;
56          int res = msgrcv(queue_id, &msg, sizeof(Msg) - sizeof(long), virtual_queue, 0);
57
58          if (res < 0) {
59              E("Failed to receive message. Maybe the queue died?");
60              break;
61          }
62
63          kill(msg.id, SIGUSR1);
64      }
65
66      return 0;
67  }
```

5.5 client/message.h File Reference

Data Structures

- struct msgbuf

  Message The structure of the message that will be sent between client and server.

Macros

- #define MAX_STRING_SIZE 1000

Typedefs

- typedef struct msgbuf Msg

  Message The structure of the message that will be sent between client and server.

5.5.1 Macro Definition Documentation

5.5.1.1 MAX_STRING_SIZE

```
#define MAX_STRING_SIZE 1000
```

Definition at line 3 of file message.h.

5.5.2 Typedef Documentation

5.5.2.1 Msg

```
typedef struct msgbuf Msg
```

Message The structure of the message that will be sent between client and server.

5.6 server/message.h File Reference

Data Structures

- struct msgbuf

  Message The structure of the message that will be sent between client and server.

Macros

- #define MAX_STRING_SIZE 1000

Typedefs

- typedef struct msgbuf Msg

  Message The structure of the message that will be sent between client and server.

5.6.1 Macro Definition Documentation

5.6.1.1 MAX_STRING_SIZE

```
#define MAX_STRING_SIZE 1000
```

Definition at line 2 of file message.h.

5.6.2 Typedef Documentation

5.6.2.1 Msg

```
typedef struct msgbuf Msg
```

Message The structure of the message that will be sent between client and server.

5.7 shutdown/message.h File Reference

Data Structures

- struct msgbuf

  Message The structure of the message that will be sent between client and server.

Macros

- #define MAX_STRING_SIZE 1000

Typedefs

- typedef struct msgbuf Msg

  Message.

5.7.1 Macro Definition Documentation

5.7.1.1 MAX_STRING_SIZE

```
#define MAX_STRING_SIZE 1000
```

Definition at line 3 of file message.h.

5.7.2 Typedef Documentation

5.7.2.1 Msg

```
typedef struct msgbuf Msg
```

Message.

The structure of the message that will be sent between client and server

5.8 HelloWorld.c File Reference

```
#include <stdio.h>
```

Functions

- int main ()

5.8.1 Function Documentation

5.8.1.1 main()

```
int main ( )
```

Definition at line 4 of file HelloWorld.c.

```
5 {
6     printf("Hello World\n");
7
8 return 0;
9 }
```

5.9 README.md File Reference

5.10 server/hypercube.c File Reference

```
#include "hypercube.h"
```

Functions

- void hc_make (int *hc)

  Initializa the structure.
- int hc_up (int idx)

  Get the index of the father of the node at the given index.
- void hc_down (int idx, int *ans)

  Get the index of the sons of the node at the given index.

5.10.1 Function Documentation

5.10.1.1 hc_down()

```
void hc_down (
            int idx,
            int * ans )
```

Get the index of the sons of the node at the given index.

Parameters

idx	the index.
ans	the output, it should be at least 2, the rest will be filled with -1.

Definition at line 27 of file hypercube.c.

```
27                                  {
28     for (int i = 0; i < 4; i++) {
29         if (idx & (1 << i)) {
30             break;
31         } else {
32             ans[i] = idx | (1 << i);
33         }
34     }
35 
36     for (int i = 0; i < 4; i++) {
37         ans[i] = (ans[i] < N) ? ans[i] : -1;
38     }
39 }
```

5.10.1.2 hc_make()

```
void hc_make (
              int * hc )
```

Initializa the structure.

Parameters

hc	an array to store the structure.

Definition at line 3 of file hypercube.c.

```
3                       {
4     for (int i = 0; i < N; i++) {
5         hc[i] = -1;
6     }
7 }
```

5.10.1.3 hc_up()

```
int hc_up (
              int idx )
```

Get the index of the father of the node at the given index.

Parameters

idx	the index.

Returns

int The index of the father

Definition at line 9 of file hypercube.c.

```
9                       {
10    if (idx) {
11        return idx  - (idx&-idx);
12    }
13
14    return N;
15 }
```

5.11 server/hypercube.h File Reference

```
#include "util.h"
```

Functions

- void hc_make (int *hc)

  Initializa the structure.
- int hc_up (int idx)

  Get the index of the father of the node at the given index.
- void hc_down (int idx, int *ans)

  Get the index of the sons of the node at the given index.

5.11.1 Detailed Description

Author

Luís Eduardo Luz Sila - 15/0137885
Claudio Segala Rodrigues Filho 15/0032552
Yan Victor dos Santos 14/0033599

Version

0.1

Date

2019-06-03

Copyright

Copyright (c) 2019

5.11.2 Function Documentation

5.11.2.1 hc_down()

```
void hc_down (
            int idx,
            int * ans )
```

Get the index of the sons of the node at the given index.

Parameters

idx	the index.
ans	the output, it should be at least 2, the rest will be filled with -1.

Definition at line 27 of file hypercube.c.

```
27                                  {
28      for (int i = 0; i < 4; i++) {
29          if (idx & (1 << i)) {
30              break;
31          } else {
32              ans[i] = idx | (1 << i);
33          }
34      }
35
36      for (int i = 0; i < 4; i++) {
37          ans[i] = (ans[i] < N) ? ans[i] : -1;
38      }
39  }
```

5.11.2.2 hc_make()

```
void hc_make (
              int * hc )
```

Initializa the structure.

Parameters

hc	an array to store the structure.

Definition at line 3 of file hypercube.c.

```
3                       {
4       for (int i = 0; i < N; i++) {
5           hc[i] = -1;
6       }
7   }
```

5.11.2.3 hc_up()

```
int hc_up (
              int idx )
```

Get the index of the father of the node at the given index.

Parameters

idx	the index.

Returns

int The index of the father

Definition at line 9 of file hypercube.c.

```
9                       {
10      if (idx) {
11          return idx  - (idx&-idx);
12      }
13
14      return N;
15  }
```

5.12 server/job.c File Reference

```
#include "util.h"
#include "job.h"
```

Functions

- Job * job_create (int id, int seconds, char *filename, int delay)

  Create a job.

5.12.1 Function Documentation

5.12.1.1 job_create()

```
Job* job_create (
            int id,
            int seconds,
            char * filename,
            int delay )
```

Create a job.

Parameters

job	id
seconds	
filename	
delay	

Returns

Job* a pointer to the node

Definition at line 4 of file job.c.

```
4                                                       {
5       Job *job = (Job*)malloc(sizeof(Job));
6
7       job->id = id;
```

```
8       job->seconds = seconds;
9       job->delay = delay;
10       job->done = false;
11       job->submission = time(NULL);
12       job->start = 0;
13       job->finish = 0;
14       job->completed = 0;
15
16       strcpy(job->filename, filename);
17
18       for (int i = 0; i < N; i++) {
19           job->node_time[i] = -1;
20           job->node_pid[i] = -1;
21       }
22
23       return job;
24   }
```

5.13 server/job.h File Reference

```
#include "util.h"
```

Data Structures

- struct job

  Um job com toda a estrutura que compõe o job.

Typedefs

- typedef struct job Job

  Um job com toda a estrutura que compõe o job.

Functions

- Job ∗ job_create (int id, int seconds, char ∗filename, int delay)

  Create a job.

5.13.1 Detailed Description

Author

Luís Eduardo Luz Sila - 15/0137885
Claudio Segala Rodrigues Filho 15/0032552
Yan Victor dos Santos 14/0033599

Version

0.1

Date

2019-06-03

Copyright

5.13.2 Typedef Documentation

5.13.2.1 Job

```
typedef struct job Job
```

Um job com toda a estrutura que compõe o job.

5.13.3 Function Documentation

5.13.3.1 job_create()

```
Job* job_create (
            int id,
            int seconds,
            char * filename,
            int delay )
```

Create a job.

Parameters

job	id
seconds	
filename	
delay	

Returns

Job* a pointer to the node

Definition at line 4 of file job.c.

```
4                                                                   {
5       Job *job = (Job*)malloc(sizeof(Job));
6
7       job->id = id;
8       job->seconds = seconds;
9       job->delay = delay;
10       job->done = false;
11       job->submission = time(NULL);
12       job->start = 0;
13       job->finish = 0;
14       job->completed = 0;
15
16       strcpy(job->filename, filename);
17
18       for (int i = 0; i < N; i++) {
19           job->node_time[i] = -1;
20           job->node_pid[i] = -1;
21       }
22
23       return job;
24   }
```

5.14 server/list.c File Reference

```
#include "list.h"
```

Functions

- Node * node_create (void *value)
  Cria um novo elemento do no.
- List * list_create ()
  Create a list.
- void list_push_back (List *list, void *value)
  Add node to the end of the list.
- void * list_pop_back (List *list)
  Remove last node.
- void list_destroy (List *list)
  Destroy and free everything in the list.

5.14.1 Function Documentation

5.14.1.1 list_create()

```
List* list_create ( )
```

Create a list.

Returns

List* a pointer to the list created.

Definition at line 18 of file list.c.

```
18                      {
19     List *list = (List*) malloc(sizeof(List));
20
21     list->sz = 0;
22     list->begin = NULL;
23     list->end = NULL;
24
25     return list;
26 }
```

5.14.1.2 list_destroy()

```
void list_destroy (
            List * list )
```

Destroy and free everything in the list.

Parameters

list	

Definition at line 71 of file list.c.

```
71                                {
72     Node *curr = list->begin;
73
74     while (curr != NULL) {
75         Node *aux = curr->nxt;
76
77         free(curr);
78
79         curr = aux;
80     }
81
82     free(list);
83     list = NULL;
84 }
```

5.14.1.3 list_pop_back()

```
void* list_pop_back (
            List * list )
```

Remove last node.

Parameters

list	a list.

Returns

void* pointer to the value of the returned

Definition at line 45 of file list.c.

```
45                                 {
46     if (list->sz) {
47         list->sz--;
48
49         Node *last = list->end;
50         Node *penultimate = last->prev;
51
52         list->end = penultimate;
53         last->prev = NULL;
54
55         if (penultimate == NULL) {
56             list->begin = NULL;
57         } else {
58             penultimate->nxt = NULL;
59         }
60
61         void* value = last->value;
62
63         free(last);
64
65         return value;
66     }
67
68     return NULL;
69 }
```

5.14.1.4 list_push_back()

```
void list_push_back (
              List * list,
              void * value )
```

Add node to the end of the list.

Parameters

list	a list.
value	a value to be stored in the node.

Definition at line 28 of file list.c.

```
28                                                  {
29     Node *new = node_create(value);
30
31     if (list->sz) {
32         Node *last = list->end;
33
34         last->nxt = new;
35         new->prev = last;
36
37         list->end = new;
38     } else {
39         list->begin = list->end = new;
40     }
41
42     list->sz++;
43 }
```

5.14.1.5 node_create()

```
Node* node_create (
              void * value )
```

Cria um novo elemento do no.

Create a node.

Parameters

value	O valor

Returns

Node*

Definition at line 8 of file list.c.

```
8                                  {
9     Node *node = (Node*) malloc(sizeof(Node));
10
11     node->prev = NULL;
```

```
12     node->nxt = NULL;
13     node->value = value;
14
15     return node;
16 }
```

5.15 server/list.h File Reference

```
#include "util.h"
#include "job.h"
```

Data Structures

- struct node

  A node. The node of a doubly linked list.

- struct list

  A list. A doubly linked list.

Typedefs

- typedef struct node Node

  A node. The node of a doubly linked list.

- typedef struct list List

  A list. A doubly linked list.

Functions

- Node * node_create (void *value)

  Create a node.

- List * list_create ()

  Create a list.

- void list_push_back (List *list, void *value)

  Add node to the end of the list.

- void * list_pop_back (List *list)

  Remove last node.

- void list_destroy (List *list)

  Destroy and free everything in the list.

5.15.1 Detailed Description

Author

Luís Eduardo Luz Sila - 15/0137885
Claudio Segala Rodrigues Filho 15/0032552
Yan Victor dos Santos 14/0033599

Version

0.1

Date

2019-06-03

Copyright

5.15.2 Typedef Documentation

5.15.2.1 List

```
typedef struct list List
```

A list. A doubly linked list.

5.15.2.2 Node

```
typedef struct node Node
```

A node. The node of a doubly linked list.

5.15.3 Function Documentation

5.15.3.1 list_create()

```
List* list_create ( )
```

Create a list.

Returns

List* a pointer to the list created.

Definition at line 18 of file list.c.

```
18                      {
19     List *list = (List*) malloc(sizeof(List));
20
21     list->sz = 0;
22     list->begin = NULL;
23     list->end = NULL;
24
25     return list;
26 }
```

5.15.3.2 list_destroy()

```
void list_destroy (
             List * list )
```

Destroy and free everything in the list.

Parameters

list	

Definition at line 71 of file list.c.

```
71                                {
72     Node *curr = list->begin;
73
74     while (curr != NULL) {
75         Node *aux = curr->nxt;
76
77         free(curr);
78
79         curr = aux;
80     }
81
82     free(list);
83     list = NULL;
84 }
```

5.15.3.3 list_pop_back()

```
void* list_pop_back (
             List * list )
```

Remove last node.

Parameters

list	a list.

Returns

void∗ pointer to the value of the returned

Definition at line 45 of file list.c.

```
45                                  {
46     if (list->sz) {
47         list->sz--;
48
49         Node *last = list->end;
50         Node *penultimate = last->prev;
51
52         list->end = penultimate;
53         last->prev = NULL;
54
55         if (penultimate == NULL) {
56             list->begin = NULL;
57         } else {
58             penultimate->nxt = NULL;
59         }
60
61         void* value = last->value;
62
63         free(last);
64
65         return value;
66     }
67
68     return NULL;
69 }
```

5.15.3.4 list_push_back()

```
void list_push_back (
              List * list,
              void * value )
```

Add node to the end of the list.

Parameters

list	a list.
value	a value to be stored in the node.

Definition at line 28 of file list.c.

```
28                                          {
29     Node *new = node_create(value);
30
31     if (list->sz) {
32         Node *last = list->end;
33
34         last->nxt = new;
35         new->prev = last;
36
```

```
37          list->end = new;
38      } else {
39          list->begin = list->end = new;
40      }
41
42      list->sz++;
43  }
```

5.15.3.5 node_create()

```
Node* node_create (
              void * value )
```

Create a node.

Parameters

value	a void pointer to the value.

Returns

Node* a pointer to the node

Create a node.

Parameters

value	O valor

Returns

Node*

Definition at line 8 of file list.c.

```
8                                   {
9       Node *node = (Node*) malloc(sizeof(Node));
10
11      node->prev = NULL;
12      node->nxt = NULL;
13      node->value = value;
14
15      return node;
16  }
```

5.16 server/torus.c File Reference

```
#include "torus.h"
```

Functions

- void tr_make (int *tr)

  Initializa the structure.
- int tr_up (int idx)

  Estrutura do no.
- void tr_down (int idx, int *ans)

  Enviar para os nós embaixo da árvore.

5.16.1 Function Documentation

5.16.1.1 tr_down()

```
void tr_down (
            int idx,
            int * ans )
```

Enviar para os nós embaixo da árvore.

Parameters

idx	
ans	

Definition at line 35 of file torus.c.

```
35                                  {
36     ans[0] = idx + M;
37
38     if (idx < M - 1) {
39         ans[1] = idx + 1;
40     }
41
42     ans[0] = ans[0] < N ? ans[0] : -1;
43     ans[1] = ans[1] < N ? ans[1] : -1;
44 }
```

5.16.1.2 tr_make()

```
void tr_make (
            int * tr )
```

Initializa the structure.

Parameters

tr	an array to store the structure.

Definition at line 3 of file torus.c.

```
3                       {
4     for (int i = 0; i < N; i++) {
5         tr[i] = -1;
6     }
7 }
```

5.16.1.3 tr_up()

```
int tr_up (
              int idx )
```

Estrutura do no.

Get the index of the father of the node at the given index.

Structure: root node 1 12 13 14 15 8 9 10 11 4 5 6 7 0 1 2 3 // First line

Definition at line 18 of file torus.c.

```
18                      {
19     if (idx) { // if isn't the root node
20         if(idx > M) {// if top line, send to first line
21             return idx - M;
22         } else {
23             return idx - 1;
24         }
25     } else { // if root node, send to scheduler
26         return N;
27     }
28 }
```

5.17 server/torus.h File Reference

```
#include "util.h"
```

Functions

- void tr_make (int *tr)

  Initializa the structure.
- int tr_up (int idx)

  Get the index of the father of the node at the given index.
- void tr_down (int idx, int ans[ ])

  Get the index of the sons of the node at the given index.

5.17.1 Detailed Description

Author

Luís Eduardo Luz Sila - 15/0137885
Claudio Segala Rodrigues Filho 15/0032552
Yan Victor dos Santos 14/0033599

Version

0.1

Date

2019-06-03

Copyright

5.17.2 Function Documentation

5.17.2.1 tr_down()

```
void tr_down (
            int idx,
            int ans[] )
```

Get the index of the sons of the node at the given index.

Parameters

idx	the index.
ans	the output, it should be at least 2, the rest will be filled with -1.

5.17.2.2 tr_make()

```
void tr_make (
            int * tr )
```

Initializa the structure.

Parameters

tr	an array to store the structure.

Definition at line 3 of file torus.c.

```
3                       {
4     for (int i = 0; i < N; i++) {
5         tr[i] = -1;
6     }
7 }
```

5.17.2.3 tr_up()

```
int tr_up (
              int idx )
```

Get the index of the father of the node at the given index.

Parameters

idx	the index.

Returns

int The index of the father

Get the index of the father of the node at the given index.

Structure: root node 1 12 13 14 15 8 9 10 11 4 5 6 7 0 1 2 3 // First line

Definition at line 18 of file torus.c.

```
18                      {
19     if (idx) { // if isn't the root node
20         if(idx > M) {// if top line, send to first line
21             return idx - M;
22         } else {
23             return idx - 1;
24         }
25     } else { // if root node, send to scheduler
26         return N;
27     }
28 }
```

5.18 server/tree.c File Reference

```
#include "tree.h"
```

Functions

- void ft_make (int *ft)

  Create a new tree structure.
- int ft_up (int idx)

  Access up values from fat tree.
- void ft_down (int idx, int *ans)

  Get the index of the sons of the node at the given index.

5.18.1 Function Documentation

5.18.1.1 ft_down()

```
void ft_down (
              int idx,
              int * ans )
```

Get the index of the sons of the node at the given index.

Parameters

idx	the index.
ans	the output, it should be at least 2, the rest will be filled with -1.

Definition at line 16 of file tree.c.

```
16                                  {
17     ans[0] = (2 * idx + 1 < N) ? 2 * idx + 1 : -1;
18     ans[1] = (2 * idx + 2 < N) ? 2 * idx + 2 : -1;
19 }
```

5.18.1.2 ft_make()

```
void ft_make (
              int * ft )
```

Create a new tree structure.

Parameters

ft	an array to start the structure

Definition at line 3 of file tree.c.

```
3                       {
4     for (int i = 0; i < N; i++) {
5         ft[i] = -1;
6     }
7 }
```

5.18.1.3 ft_up()

```
int ft_up (
              int idx )
```

Access up values from fat tree.

Parameters

idx	index

Returns

int

Definition at line 9 of file tree.c.

```
9                       {
10   if (idx)
11       return (idx - 1) / 2;
12
13   return N;
14 }
```

5.19 server/tree.h File Reference

```
#include "util.h"
```

Functions

- void ft_make (int *ft)

  Create a new tree structure.
- int ft_up (int idx)

  Access up values from fat tree.
- void ft_down (int idx, int *ans)

  Get the index of the sons of the node at the given index.

5.19.1 Function Documentation

5.19.1.1 ft_down()

```
void ft_down (
            int idx,
            int * ans )
```

Get the index of the sons of the node at the given index.

Parameters

idx	the index.
ans	the output, it should be at least 2, the rest will be filled with -1.

Definition at line 16 of file tree.c.

```
16                                  {
17     ans[0] = (2 * idx + 1 < N) ? 2 * idx + 1 : -1;
18     ans[1] = (2 * idx + 2 < N) ? 2 * idx + 2 : -1;
19 }
```

5.19.1.2 ft_make()

```
void ft_make (
              int * ft )
```

Create a new tree structure.

Parameters

ft	an array to start the structure

Definition at line 3 of file tree.c.

```
3                       {
4     for (int i = 0; i < N; i++) {
5         ft[i] = -1;
6     }
7 }
```

5.19.1.3 ft_up()

```
int ft_up (
              int idx )
```

Access up values from fat tree.

Parameters

idx	index

Returns

int

Definition at line 9 of file tree.c.

```
9                       {
10   if (idx)
11       return (idx - 1) / 2;
12
13   return N;
14 }
```

5.20 server/util.c File Reference

```
#include "util.h"
```

Functions

- bool try_cast_int (char *num, int *result)

5.20.1 Function Documentation

5.20.1.1 try_cast_int()

```
bool try_cast_int (
            char * num,
            int * result )
```

Definition at line 3 of file util.c.

```
3                                               {
4     int sz = strlen(num);
5
6     for (int i = sz - 1, n = 1; i >= 0; i--, n *= 10) {
7         if (num[i] >= '0' && num[i] <= '9') {
8             (*result) += n * (num[i] - '0');
9         } else {
10             return false;
11         }
12     }
13
14     return true;
15 }
```

5.21 server/util.h File Reference

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <signal.h>
```

Macros

- #define RED "\033[0;31m"
- #define GREEN "\033[0;32m"
- #define YELLOW "\033[0;33m"
- #define RESET "\033[0m"
- #define MSG_FLAG 0x123
- #define TREE 1
- #define HYPERCUBE 2
- #define TORUS 3
- #define S(x) printf("%s%s%s\n", GREEN, x, RESET); fflush(stdout);
- #define E(x) printf("%s%s%s\n", RED, x, RESET); fflush(stdout);
- #define KEY 15003
- #define N 15
- #define M 4
- #define PATH "../prog/"

Functions

- y_cast_int (char ∗string, int ∗inteiro)

  Converte char em string.

5.21.1 Detailed Description

Author

Luís Eduardo Luz Sila - 15/0137885
Claudio Segala Rodrigues Filho 15/0032552
Yan Victor dos Santos 14/0033599

Version

0.1

Date

2019-06-03

Copyright

5.21.2 Macro Definition Documentation

5.21.2.1 E

```
#define E(
             x ) printf("%s%s%s\n", RED, x, RESET); fflush(stdout);
```

Definition at line 40 of file util.h.

5.21.2.2 GREEN

```
#define GREEN "\033[0;32m"
```

Definition at line 29 of file util.h.

5.21.2.3 HYPERCUBE

```
#define HYPERCUBE 2
```

Definition at line 36 of file util.h.

5.21.2.4 KEY

```
#define KEY 15003
```

Definition at line 42 of file util.h.

5.21.2.5 M

```
#define M 4
```

Definition at line 44 of file util.h.

5.21.2.6 MSG_FLAG

```
#define MSG_FLAG 0x123
```

Definition at line 33 of file util.h.

5.21.2.7 N

```
#define N 15
```

Definition at line 43 of file util.h.

5.21.2.8 PATH

```
#define PATH "../prog/"
```

Definition at line 46 of file util.h.

5.21.2.9 RED

```
#define RED "\033[0;31m"
```

Definition at line 28 of file util.h.

5.21.2.10 RESET

```
#define RESET "\033[0m"
```

Definition at line 31 of file util.h.

5.21.2.11 S

```
#define S(
             x ) printf("%s%s%s\n", GREEN, x, RESET); fflush(stdout);
```

Definition at line 39 of file util.h.

5.21.2.12 TORUS

```
#define TORUS 3
```

Definition at line 37 of file util.h.

5.21.2.13 TREE

```
#define TREE 1
```

Definition at line 35 of file util.h.

5.21.2.14 YELLOW

```
#define YELLOW "\033[0;33m"
```

Definition at line 30 of file util.h.

5.21.3 Function Documentation

5.21.3.1 y_cast_int()

```
y_cast_int (
            char * string,
            int * inteiro )
```

Converte char em string.

Parameters

string	passa a string para ser transformada em inteiro
inteiro	passa o inteiro que receberá a string

5.22 util/semaphore.c File Reference

```
#include "semaphore.h"
```

Data Structures

- struct semaphore

Functions

- void sem_op (Semaphore *sem, int idx, int num, int op, int flag)
  Get the index of the father of the node at the given index.
- void P (Semaphore *sem)
  Increase the semaphore.
- void V (Semaphore *sem)
  Decrease the semaphore.

5.22.1 Function Documentation

5.22.1.1 P()

```
void P (
            Semaphore * sem )
```

Increase the semaphore.

Parameters

sem	the semaphore.

Definition at line 14 of file semaphore.c.

```
14                          {
15     sem_op(sem, 0, 0, 0, 0);
16     sem_op(sem, 1, 0, 1, 0);
17
18     int res = semop(sem->id, sem->def, 2);
19
20     if (res < 0) {
21         printf("Erro no p=%d\n", errno);
22     }
23 }
```

5.22.1.2 sem_op()

```
void sem_op (
            Semaphore * sem,
            int idx,
            int num,
            int op,
            int flag )
```

Get the index of the father of the node at the given index.

Parameters

sem	the semaphore.
idx	the index.
num	the number of operations.
op	the operation.
flag	the flag.

Definition at line 8 of file semaphore.c.

```
8                                                       {
```

```
9    sem->def[idx].sem_num = num;
10    sem->def[idx].sem_op = op;
11    sem->def[idx].sem_flg = flag;
12 }
```

5.22.1.3 V()

```
void V (
            Semaphore * sem )
```

Decrease the semaphore.

Parameters

sem	the semaphore.

Definition at line 25 of file semaphore.c.

```
25                          {
26     sem_op(sem, 0, 0, -1, 0);
27
28     int res = semop(sem->id, sem->def, 2);
29
30     if (res < 0) {
31         printf("Erro no p=%d\n", errno);
32     }
33 }
```

5.23 util/semaphore.h File Reference

```
#include "util.h"
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

Typedefs

- typedef struct semaphore Semaphore

Functions

- void sem_op (Semaphore *sem, int idx, int num, int op, int flag)

  Get the index of the father of the node at the given index.
- void P (Semaphore *sem)

  Increase the semaphore.
- void V (Semaphore *sem)

  Decrease the semaphore.

5.23.1 Typedef Documentation

5.23.1.1 Semaphore

```
typedef struct semaphore Semaphore
```

Definition at line 10 of file semaphore.h.

5.23.2 Function Documentation

5.23.2.1 P()

```
void P (
            Semaphore * sem )
```

Increase the semaphore.

Parameters

sem	the semaphore.

Definition at line 14 of file semaphore.c.

```
14                          {
15     sem_op(sem, 0, 0, 0, 0);
16     sem_op(sem, 1, 0, 1, 0);
17
18     int res = semop(sem->id, sem->def, 2);
19
20     if (res < 0) {
21         printf("Erro no p=%d\n", errno);
22     }
23 }
```

5.23.2.2 sem_op()

```
void sem_op (
            Semaphore * sem,
            int idx,
            int num,
            int op,
            int flag )
```

Get the index of the father of the node at the given index.

Parameters

sem	the semaphore.
idx	the index.
num	the number of operations.
op	the operation.
flag	the flag.

Definition at line 8 of file semaphore.c.

```
8                                                                   {
9   sem->def[idx].sem_num = num;
10   sem->def[idx].sem_op = op;
11   sem->def[idx].sem_flg = flag;
12 }
```

5.23.2.3 V()

```
void V (
            Semaphore * sem )
```

Decrease the semaphore.

Parameters

sem	the semaphore.

Definition at line 25 of file semaphore.c.

```
25                          {
26     sem_op(sem, 0, 0, -1, 0);
27
28     int res = semop(sem->id, sem->def, 2);
29
30     if (res < 0) {
31         printf("Erro no p=%d\n", errno);
32     }
33 }
```

Index