/\*

\* Brief: Source file of the main program 'parque'

\* Autors: Catarina Ramos, Mário Fernandes;

\* Class: 5

\* Group: 14

\*/

#include "resources.h"

static int closing\_park;

static int n\_spaces\_occupied;

static int n\_spaces\_total;

static clock\_t opening\_time; //since the opening of the park

static int file\_parque; //file descriptor of parque.log

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER; //sync of entrance/exit and number of spaces\_ocupied

/\*

\*To know if the controllers fifo was opened (in case the generator wasn't executed

\* the fifo controller blocks until the fifo is opened elsewhere)

\*/

static int fifoN\_opened = 0;

static int fifoS\_opened = 0;

static int fifoE\_opened = 0;

static int fifoW\_opened = 0;

/\*

\* THREAD DETACHED ATTENDANT (ARRUMADOR)

\* Attends a car entrance until he leaves the park.

\*/

void \* attendant(void \*argc){

struct generator\_info \* car = (struct generator\_info \*) argc;

char info[100];

int fifo\_car;

char fifo\_pathname[16];

clock\_t current\_time;

char state[16]; //c - close , e - enter , x - exit , f - full

char st\_code;

//CONNECT TO PRIVATE CAR FIFO

sprintf(fifo\_pathname,"fifo%d",car->id\_car);

if((fifo\_car = open(fifo\_pathname,O\_WRONLY)) == -1) //NON\_BLOCK ?

error("Could not open fifo car\n");

/\*

\* CRITIC SESSION ----------------------------------------

\*/

if(pthread\_mutex\_lock(&mutex) != 0)

error("Could not lock mutex\n");

//CHECK PARK CONDITION

if(n\_spaces\_total-n\_spaces\_occupied > 0 && !closing\_park){

st\_code = 'e';

sprintf(state,"estacionamento");

n\_spaces\_occupied++;

}

else if(closing\_park){

st\_code = 'c';

sprintf(state,"encerrado");

}

else if (n\_spaces\_total-n\_spaces\_occupied <= 0 && !closing\_park){

st\_code = 'f';

sprintf(state,"cheio!");

}

//TIME OF REQUEST RECEIVED (ENTER)

if ((current\_time = clock()) == -1)

error("Couldn't start clock\n");

current\_time -= opening\_time;

//WRITE PARK STATUS

sprintf(info," %16d ; %13d ; %15d ; %15s \n",(int)current\_time,n\_spaces\_occupied,car->id\_car,state);

write(file\_parque,info,strlen(info));

if(pthread\_mutex\_unlock(&mutex) != 0)

error("Could not unlock mutex\n");

/\*

\* END CRITIC SESSION ----------------------------------------

\*/

//INFORM FIFO CAR ABOUT PARK CONDITION

if(write(fifo\_car,&st\_code,sizeof(st\_code)) == -1)

error("Could not write on fifo car\n");

//PARK FULL OR CLOSED

if(st\_code == 'f' || st\_code == 'c'){

close(fifo\_car);

free(car);

return NULL;

}

//PARKING TIME (TIMER)

pthread\_t t;

pthread\_create(&t,NULL,timer,&car->parking\_time);

pthread\_join(t,NULL);

// TIME OF LEAVING

if ((current\_time = clock()) == -1)

error("Couldn't start clock\n");

current\_time -= opening\_time;

// INFORM FIFO CAR TO LEAVE

st\_code = 'x'; //exit - x

if(write(fifo\_car,&st\_code,sizeof(st\_code)) == -1)

error("Could not write on fifo car\n");

/\*

\* CRITIC SESSION ----------------------------------------

\*/

if(pthread\_mutex\_lock(&mutex) != 0)

error("Could not lock mutex\n");

n\_spaces\_occupied--;

//WRITE PARK STATUS

sprintf(info," %16d ; %13d ; %15d ; %15s \n",(int)current\_time,n\_spaces\_occupied,car->id\_car,"saida");

if(write(file\_parque,info,strlen(info)) == -1)

error("Could not write on parque.log\n");

if(pthread\_mutex\_unlock(&mutex) != 0)

error("Could not unlock mutex\n");

/\*

\* END CRITIC SESSION ----------------------------------------

\*/

close(fifo\_car);

free(car);

return NULL;

}

//------------------------------------------------------------------------------------------

/\*

\* THREAD CONTROLLER (controlador)

\* Controls the entrance and exit of cars to the park.

\* When the park closes, it doesn't answer for more requests.

\*/

void \* controller(void \* argc){

char entrance = \*(char \*) argc;

char fifo\_pathname[16];

int fifo;

int err;

struct generator\_info car\_info;

pthread\_t attendant\_thread;

pthread\_attr\_t attr;

int n\_read;

//FOR DETACHED THREAD

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_DETACHED);

/\*

\* FIFO CONTROLER (GENERIC)

\*/

sprintf(fifo\_pathname,"fifo%c",entrance);

if((fifo = open(fifo\_pathname,O\_RDONLY | O\_CREAT,0666)) == -1)

error("Could not open fifo\n");

if(strcmp(fifo\_pathname,"fifoN") == 0)

fifoN\_opened = 1;

if(strcmp(fifo\_pathname,"fifoS") == 0)

fifoS\_opened = 1;

if(strcmp(fifo\_pathname,"fifoE") == 0)

fifoE\_opened = 1;

if(strcmp(fifo\_pathname,"fifoW") == 0)

fifoW\_opened = 1;

/\*

\* RECEIVE CARS REQUESTS

\*/

while(closing\_park == 0){

if((n\_read = read(fifo,&car\_info,sizeof(struct generator\_info))) == -1)

error("Could not read from fifo controller\n");

else if(n\_read > 0){

struct generator\_info \* car = malloc(sizeof(struct generator\_info));

\*car = car\_info;

if((err = pthread\_create(&attendant\_thread, &attr,attendant,car)) != 0)

error("Pthread\_create erro for north\_thread");

}

}

close(fifo);

unlink(fifo\_pathname);

pthread\_exit(NULL);

}

//------------------------------------------------------------------------------------------

/\*

\* MAIN THREAD PARQUE

\*/

int main(int argc , char \*argv[]) {

double time;

pthread\_t north\_thread, south\_thread , west\_thread , east\_thread;

clock\_t start\_time, end\_time;

closing\_park = 0;

n\_spaces\_occupied = 0;

char info[256];

//Checks the number of arguments

if(argc != 3)

error("Wrong number of arguments\n");

//Checks if the arguments are valid

if((n\_spaces\_total = atoi(argv[1])) == 0 || (time = atoi(argv[2])) == 0)

error("Invalid Arguments, must be an integer!\n");

/\*

\* PARQUE.LOG

\*/

if ((file\_parque = open("parque.log", O\_WRONLY | O\_CREAT | O\_TRUNC, 0644)) == -1) {

error("Couldn't open parque.log.\n");

}

sprintf(info," t(ticks) ; n\_lug ; id\_viat ; observ \n");

write(file\_parque,info,strlen(info));

/\*

\* SEMAPHORE (PARQUE<=>GERADOR)

\*/

sem\_t \* semaphore = malloc(sizeof(sem\_t));

if((semaphore = sem\_open("semaphore", O\_CREAT | O\_EXCL, 0666, 1)) == SEM\_FAILED){

if(errno == EEXIST){

if((semaphore = sem\_open("semaphore", 0)) == SEM\_FAILED) {

error("Couldn't create semaphore\n");

}

}

else

error("Couldn't create semaphore\n");

}

/\*

\* FIFO CONTROLLERS

\*/

if(mkfifo("fifoN",0666) == -1)

error("fifo controller could not be created!\n");

if(mkfifo("fifoS",0666) == -1)

error("fifo controller could not be created!\n");

if(mkfifo("fifoE",0666) == -1)

error("fifo controller could not be created!\n");

if(mkfifo("fifoW",0666) == -1)

error("fifo controller could not be created!\n");

/\*

\* THREADS CONTROLLER

\*/

if(pthread\_create(&north\_thread, NULL,controller, "N") != 0)

error("Pthread\_create erro for north\_thread");

if(pthread\_create(&south\_thread, NULL,controller, "S") != 0)

error("Pthread\_create erro for south\_thread");

if(pthread\_create(&west\_thread, NULL,controller, "W") != 0)

error("Pthread\_create erro for west\_thread");

if(pthread\_create(&east\_thread, NULL,controller, "E") != 0)

error("Pthread\_create erro for east\_thread");

/\*

\* CLOCKS

\*/

end\_time = time\*CLOCKS\_PER\_SEC;

if ((start\_time = clock()) == -1) //Start clock

error("Couldn't start clock\n");

opening\_time = start\_time;

/\*

\* TIMER PARK OPEN

\*/

pthread\_t t;

pthread\_create(&t,NULL,timer,&end\_time);

pthread\_join(t,NULL);

/\*

\* CRITIC SESSION ---------------------------------------

\*/

if(sem\_wait(semaphore) == -1)

error("Wait for semaphore error\n");

closing\_park = 1;

if(sem\_post(semaphore) == -1)

error("Post semaphore error\n");

/\*

\* END CRITIC SESSION ---------------------------------------

\*/

/\*

\* IF the controllers fifo weren't opened in generator then the program will wait for it's opening

\* This way we can controll if the fifos were opened in both sides and skip in case they just opened in park

\*/

if(fifoN\_opened == 0){

pthread\_cancel(north\_thread);

unlink("fifoN");

}

else

pthread\_join(north\_thread,NULL);

if(fifoS\_opened == 0){

pthread\_cancel(south\_thread);

unlink("fifoS");

}

else

pthread\_join(south\_thread,NULL);

if(fifoE\_opened == 0){

pthread\_cancel(east\_thread);

unlink("fifoE");

}

else

pthread\_join(east\_thread,NULL);

if(fifoW\_opened == 0){

pthread\_cancel(west\_thread);

unlink("fifoW");

}

else

pthread\_join(west\_thread,NULL);

sem\_unlink("semaphore");

pthread\_exit(0);

}