/\*

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

\* Autors: Catarina Ramos, Mário Fernandes;

\* Class: 5

\* Group: 14

\*/

#include "resources.h"

static clock\_t function\_time; //function start time of generator

static int file\_gerador; //file descriptor of gerador.log

static sem\_t \*semaphore; //to sync with park

/\*

\* THREAD DETACHED CAR (VIATURA)

\* This thread communicates with the controller car\_info->destinatio fifo and sends a request

\* to enter the park. It waits for the answer via the private fifo of the car. If the access to the park is denied,

\* because it's full or closed, the thread terminates, otherwise, the thread waits for another answer from the controller

\* so it can exit the park.

\* The car\_info->duration represents the time since the creation of the struct that represents the car until the

\* exit of the park.

\* After receiving an answer form the controller fifo, is written on gerador.log the car status.

\* The car private fifo name is FIFOID, for example: FIFO16 (where ID = 16);

\*/

void \* car(void \* argc){

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

int fifo\_dest; //controller fifo

char fifo\_pathname[36];

char answer = ' '; //answer from controller

int fifo\_car;

int n\_read = 0;

clock\_t current\_time;

char info[256];

/\*

\* CRITIC SESSION ------------------------------------ SYNC WITH PARK TO KNOW IF IT IS CLOSED

\*/

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

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

//ATTEMPT TO CONNECT TO CONTROLLER FIFO

sprintf(fifo\_pathname,"fifo%c",car\_info->destination);

if((fifo\_dest = open(fifo\_pathname,O\_WRONLY)) == -1){

if(errno != ENOENT)

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

//IF CONSTROLLER FIFO IS CLOSED/DOESN'T EXIST

//WRITE INFO ON GERADOR.LOG

sprintf(info," %16d ; %14d ; %4c ; %14d ; %16s ; %16s\n",(int)car\_info->time,(int)car\_info->id\_car,car\_info->destination,(int)car\_info->parking\_time, "?","encerrado");

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

error("Problem on writing on file\n");

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

error("Post semaphore error\n");

/\*

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

\*/

free(car\_info);

return NULL;

}

//MAKE PRIVATE FIFO

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

if(mkfifo(fifo\_pathname,0666) != 0)

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

//SEND REQUEST TO ACCESS PARK (and open the private fifo there)

if(write(fifo\_dest,car\_info,sizeof(struct generator\_info)) == -1)

error("Error on writting to fifo\_dest\n");

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

error("Post semaphore error\n");

/\*

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

\*/

//OPEN PRIVATE FIFO

if((fifo\_car = open(fifo\_pathname,O\_RDONLY)) == -1)

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

//WAIT ANSWER FROM CONTROLLER TO ENTER

while((n\_read = read(fifo\_car,&answer,sizeof(char))) == 0){

}

switch (answer) {

case 'e': {

sprintf(car\_info->state, "entrada");

break;

}

case 'f': {

sprintf(car\_info->state, "cheio!");

break;

}

case 'c': {

sprintf(car\_info->state, "encerrado");

break;

}

case 'x': {

sprintf(car\_info->state, "saida");

break;

}

default: {

error("wrong answer\n");

}

}

//WRITE CAR STATUS

sprintf(info," %16d ; %14d ; %4c ; %14d ; %16s ; %16s\n",(int)car\_info->time,(int)car\_info->id\_car,car\_info->destination,(int)car\_info->parking\_time, "?",car\_info->state);

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

error("Problem on writing on file\n");

//TERMINATE THREAD IF PARK = CLOSED | FULL

if(answer == 'f' || answer == 'c'){

close(fifo\_car);

unlink(fifo\_pathname);

free(car\_info);

return NULL;

}

//WAIT FOR EXIT INSTRUCTION FROM CONTROLLER

n\_read = 0;

while((n\_read = read(fifo\_car,&answer,sizeof(char))) == 0 || answer != 'x'){ //'x' - exit

}

if(n\_read == -1)

error("Error on getting 2 answer from private car fifo\n");

//CALCULATES CAR DURATION (LIFE TIME)

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

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

car\_info->duration = current\_time - function\_time;

car\_info->time = car\_info->duration + car\_info->time;

//WRITE CAR STATUS

sprintf(info," %16d ; %14d ; %4c ; %14d ; %16d ; %16s\n",(int)car\_info->time,(int)car\_info->id\_car,car\_info->destination,(int)car\_info->parking\_time, (int)car\_info->duration,"saida");

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

error("Problem on writing on file\n");

close(fifo\_car);

unlink(fifo\_pathname);

free(car\_info);

return NULL;

}

/\*

\* MAIN THREAD GERADOR

\* Thread that generates cars for a certain time. The time between cars generations are 50% 0 ticks, 30% min\_time ticks

\* or 20% 2\*min\_time ticks.

\* The car generator generates random information about the car and creates a thread car where the previous

\* information.

\*/

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

int time\_generator;

clock\_t min\_time;

clock\_t start\_time, curr\_time, end\_time;

pthread\_t new\_car;

pthread\_attr\_t attr;

srand(time(NULL));

int ID = 1;

pthread\_t t; //timer

char info[256];

//Checks the number of arguments

if (argc != 3)

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

//Checks if the arguments are valid

if ((time\_generator = atoi(argv[1])) == 0 || (min\_time = atoi(argv[2])) == 0)

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

/\*

\* GERADOR.LOG

\*/

if ((file\_gerador = open("gerador.log", O\_WRONLY | O\_CREAT | O\_TRUNC, 0666)) == -1) {

error("Couldn't open gerador.log on gerador.main\n");

}

sprintf(info," t(ticks) ; id\_car ; dest ; t\_parking ; duration ; observ \n");

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

error("Problem on writting on file\n");

/\*

\* SEMAPHORE (PARQUE<=>GERADOR)

\*/

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");

}

/\*

\* CLOCKS

\*/

end\_time = CLOCKS\_PER\_SEC \* time\_generator; //time\_generator (seconds) in clock ticks

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

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

function\_time = start\_time;

if ((curr\_time = clock()) == -1) //calculates the current time

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

curr\_time -= start\_time;

//FOR DETACHED THREADS CAR

pthread\_attr\_init(&attr);

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

//START GENERATING CARS

while (curr\_time < end\_time) {

struct generator\_info \*info\_car = malloc(sizeof(struct generator\_info));

//RANDOM INFO

info\_car->id\_car = ID++;

int i = rand() % 4; //controller (N/S/W/E)

switch (i) {

case 0:

info\_car->destination = 'N';

break;

case 1:

info\_car->destination = 'S';

break;

case 2:

info\_car->destination = 'W';

break;

case 3:

info\_car->destination = 'E';

break;

}

info\_car->parking\_time = (rand() % 10 + 1) \* min\_time;

info\_car->duration = -1; //undefined

sprintf(info\_car->state," ");

//IF TIME HAS ENDED THEN BREAK

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

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

curr\_time -= start\_time;

if (curr\_time >= end\_time)

break;

info\_car->time = curr\_time;

//CREATE DETACHED THREAD CAR

if (pthread\_create(&new\_car, &attr, car, info\_car) != 0)

error("Pthread\_create erro for new\_car");

//RANDOM TIME BETWEEN TWO CARS GENERATIONS

clock\_t \* wait\_time = malloc(sizeof(clock\_t));

\*wait\_time = rand() % 10 + 1;

if (\*wait\_time <= 5) //50% , mult 0x

\*wait\_time = 0;

else if (\*wait\_time > 5 && \*wait\_time <= 8) //30% , mult 1x

\*wait\_time = min\_time;

else //20% , mult 2x

\*wait\_time = 2 \* min\_time;

//TIMER

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

pthread\_join(t,NULL);

free(wait\_time);

}

sem\_unlink("semaphore");

pthread\_exit(0);

}