PL1

11. Write a program that initializes an array numbers with 1000 random integers in the range [0,255]. The program should:

a) create 5 child processes that will concurrently find the maximum value of 1/5 of the array.

b) after computing the maximum value of the entire array, the parent process should create one child process to perform the following calculation result[i]=((int) numbers[i]/max\_value)\*100 on half of the array and print the result.

c) the parent process should perform the same calculation on the other half of the array.

d) both child and parent process must perform the computation concurrently, but the output must be sorted by

the array indexes.

12. Write a function spawn\_childs(int n) which generates n child processes. a) Ensure that the function returns to the child process an index (1 for the first child process created, 2 for the second, 3 for the third and so on) and that to the parent process returns 0. b) Write a program that uses the function spawn\_childs() to create 6 child processes. Each child process should exit returning its index value \* 2. The parent should wait for all processes created.

PL2

13. Write a program that creates a child process. • Workflow of the parent process. The parent executes task A for 3 seconds. Once task A finishes, the parent displays on the screen the message “Task A: done!” and notifies the child sending it a SIGUSR1 signal. Finally, the parent waits for the child to complete, to display the message “Job is complete!”. • Workflow of the child process. The child executes task B for a random amount of time, between 1 to 5 seconds. Once task B completes, the child displays on the screen the message “Task B: done!”. Once tasks A and B are complete, the child will execute task C for 1 second. When task C is complete, the child displays the message “Task C: done!” and exits.

16. A software house wants to evaluate the potential of a new algorithm to process data in parallel to extract valuable information in a faster way than the solution in production. For this purpose, this algorithm will be tested in a simulated application that uses parallel/concurrent programming. You should develop a program where the master process (i.e. the parent) creates 50 worker processes (i.e. children).

• Workflow of the worker processes. Each worker process starts executing of the simulate1() function. This function returns 1 (one) if it finds relevant data or 0 (zero) if not.

When this function completes, the worker process sends the parent process a SIGUSR1 signal in the case of sucess or a SIGUSR2 signal otherwise.

After that, the process will wait for a signal from the master process before starting to execute simulate2() function.

• Workflow of the master process. The master process waits for the notifications sent by the workers.

After 25 processes have finished, the master process will take one of two possible actions:

(a) If none of the reported searches are successful, the parent process will display the message “Inefficient algorithm!” and terminate all children.

(b) Otherwise, the master process will send the SIGUSR1 signal to all the worker processes. Every worker process will immediately start the execution of the simulate2() function, even if it means interrupting the simulate1() function. Use randomly generated values to simulate the execution time and the result of both simulate1() and simulate2() functions. Set a very low probability of success for the simulate1() function to ensure both scenarios will occur.

PL2

13. Consider a factory line composed of 4 machines and a storage area

a. Machine M1 cuts 5 pieces, 5 pieces at a time, transfers the pieces to M2 and notifies M2;

b. Machine M2, folds the pieces, 5 at a time, transfers the pieces to M3 and notifies M3;

c. Machine M3 welds the pieces, 10 pieces at a time, transfers the pieces to machine M4 and notifies M4;

d. Machine M4 packs 100 pieces at a time, transfer them to storage A1 and adds the produced parts to the inventory. Write a program which uses processes to represent machines, pipes to represent the flux of pieces. The program should simulate the production of 1000 pieces. All operations must be correctly printed on screen.

PL3

9. Implement a program that creates a new process. One will be the producer and the other the consumer. Among them should be created a shared memory area with a circular buffer to store ten integers and the necessary synchronization variables in a shared memory area.

The producer puts increasing values in the buffer that should be printed by the consumer. Consider that thirty values are exchanged between them. The consumer can read elements from the buffer if the number of elements is bigger than 0.

10. Solve exercise 9 again, performing synchronization accessing the shared memory region, supported by unnamed pipes.

Suggestion: you can use a pipe where you place 10 integers/chars. This pipe will control the producer’s access to the shared memory region, which in the case of already having 10 integers in shared memory, will have to wait passively. Use a second pipe to synchronize the consumer’s access to the shared memory region. In the case of no values to consume, the consumer process should wait passively.

PL4

14. Consider a shared memory area (e.g. containing 10 integers) which allows only a one-way communication between a set of processes Ax and a set of processes Bx. Only one process, at a time, from Ax or Bx, is allowed to place its info on the shared memory. After one second the process should signal that other processes, from the same set, can place content on the shared memory. If there are no other processes ready to access the shared memory, then a process from the other set of processes can gain access to the shared memory. Demonstrate the operation of such a program by simulating a set of 3 Ax processes and a set of 2 Bx processes, each of them producing data (10 integers) every 5 seconds, and processes from Bx producing data every 6 seconds. Make sure your printing on screen clearly show the execution of the different steps in this program.

16. From an initial vector of 10000 integers randomly chosen in the range 1-10000, it is intended to fill a final vector of 1000 positions with the moving average of 10 values and determine the largest value found (on the final vector). To resolve the problem, design a C program with the following requirements: (a) Five child processes (P1 to P5) must be created that fill the final vector according to the conditions stated above. (b) Each child process must process 1/5th of the vector. (c) Create another child process (Pmax) that, running in parallel with the two processes mentioned above, should print a warning message whenever a new larger value is found in the final vector. This process should not use active waiting. (d) The final vector and value of the largest value should be printed by the parent process after being sure that P1-5 have finished processing, correctly.

PL5

4. Implement a program to multiplies two matrices. The program should: • Use two matrices, each with 8x8 dimensions; • Create two (2) threads to fill the two matrices; • Create eight (8) more threads to solve the problem (perform the matrices multiplication); 1 SCOMP - LEI Threads POSIX • The main thread should wait for all threads to end and then print out the calculated matrix, which should be stored on a global variable.

6. Implement a program that outputs prime numbers. It should begin by asking the user for a highest positive value. Then a created thread will outputs all the prime numbers that are less than or equal to the number entered by the user.

7. ”Totoloto” is a lottery game which encompasses two separate lotteries:

• First is a pick-5 from 49 numbers (for our exercise only this part will be relevant);

• Second is a pick-1 from 13 numbers. To play the game, the gambler must choose 5 numbers from 1 to 49 and 1 number from 1 to 13 by marking the numbered squares on a bet slip. Then the bet slip is registered by a lottery retailer (or agent) that enters the selection in the on-line terminal, to produce a game ticket. A draw will occur in which a ”key” is drawn.

9. Implement a railway simulator. The railway network’s infrastructure is composed by four city train stations (”City A”, ”City B”, ”City C” and ”City D”) and the connections are as showed in Figure 1. T

10. A retail market study collected a set of data related to 5 products sold in 3 hypermarkets. The data collected is stored in an array, called Vec with a size of 10000. Each element of the array is a triple {id h, id p, p}, where:

• id h hypermarket identifier;

• id p product identifier;

• p product price.

Implement a program to compute which hypermarket has the lowest sum for the 5 products. The program should:

• Create 6 threads (T1, T2, T3, T4, T5 and T6);

• Split the computation into 3 stages: filtering, computing and presentation;

• For the filtering stage it should:

– Use 3 threads (T1, T2 and T3);

– For each thread read data from Vec and write on a specific array associated with the hypermarket identifier (id h);

– Associate an hypermarket identifier (id h) with a specific array, i. e., for the first identifier the Vec1 array should be used, Vec2 array for the second identifier and Vec3 array for the third hypermarket identifier (see Figure 2);

– Signalize that all 3 threads ended the filtering stage.

• For the computing stage:

– It should start after all filtering threads (T1, T2, T3) have finished;

– Upon being signaled the end of the filtering stage, threads T4, T5 and T6 start its execution;

– Each thread processes only the data of one hypermarket (identifier), i.e. T4 for Vec1, T5 for Vec2 and T6 for Vec3;

– To determine the cost of the 5 products, the threads should:

∗ Compute the average price of each product;

∗ Sum all 5 average prices. – A global variable should be updated with the lowest cost as well as the hypermarket identifier. The presentation stage is performed on the main thread that prints out values for both cost and hypermarket identifier. Note that in this thread no computation is executed as it only presents the information.

PL1 EX11

#include <stdio.h> #include <unistd.h> #include <stdlib.h> #include <sys/types.h> #include <time.h> #include <wait.h>

#define ARRAY\_SIZE 1000

#define CHILDREN 5

int main(){

int array[ARRAY\_SIZE];

time\_t t;

float max = 0;

srand((unsigned)time(&t));

int i;

int n = rand() % 255;

for (i = 0; i < ARRAY\_SIZE; i++){

array[i] = rand() % 255;}

for (i = 0; i < CHILDREN; i++)

{pid\_t p = fork();

if (p == 0) {

for (int k = i \* 200; k < (i + 1) \* 200; k++){

if (array[k] > max) { max = array[k]; } }

exit(max); } }

int aux; int status;

for(i = 0; i < CHILDREN; i++){

aux = wait(&status);

if (WEXITSTATUS(status) > max) {

max = WEXITSTATUS(status); }

}

float result[ARRAY\_SIZE];

pid\_t p\_child = fork();

if (p\_child == 0) {

for (i = 0; i < ARRAY\_SIZE / 2; i++){

result[i] = ((int)array[i] / max) \* 100;

printf("Result[%d] = %lf\n", i, result[i]); }

exit(1);

}else{wait(NULL);

for (int n = ARRAY\_SIZE / 2; n < ARRAY\_SIZE; n++){result[n] = ((int)array[n] / max) \* 100; printf("Result[%d] = %lf\n", n, result[n]); }}}

PL1 EX12

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

void spawn\_childs(int n){

int i;

pid\_t a;

int status;

for (i = 0; i < n; i++){

a = fork();

if (a == 0) {

exit(i + 1);

} }

for(i = 0; i < n; i++){

wait(&status);

printf("ID = %d \n", WEXITSTATUS(status)); }

printf("ID = 0\n");}

int main(){

int n = 6;

spawn\_childs(n);

return 0;

}

PL1B EX13

#include <stdio.h>#include <sys/types.h>#include <unistd.h>#include <stdlib.h>#include <sys/wait.h> #include<time.h>#include <signal.h>#include <string.h>

volatile sig\_atomic\_t flag = 0;

void handle\_SIG(int signo) {

char msg[80];flag = 1;

sprintf(msg, "Job is complete!\n");

write(STDOUT\_FILENO, msg, strlen(msg)); }

int main(void) {

int state;

time\_t t;

srand((unsigned)time(&t));

struct sigaction act;

sigemptyset(&act.sa\_mask); /\* No signals blocked \*/

act.sa\_handler = handle\_SIG;

sigaction(SIGUSR1, &act, NULL);

pid\_t pid = fork();

if (pid > 0){

printf("Task A: started!\n");

sleep(3);

printf("Task A: done!\n");

kill(pid, SIGUSR1);

waitpid(pid, &state, 0);

printf("The job is complete!\n"); }

else{

while (flag != 1) {

pause();

}int randomNumber = rand() % 6;

printf("Task B: started!\n");

sleep(randomNumber);

printf("Task B: done!\n");

printf("Task C: started!\n");

printf("Task C: done!\n");exit(0);

} }

PL1B EX16

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

#include <time.h> #include <sys/wait.h>

#define NUM\_WORKERS 50 #define SUCCESS\_PROBABILITY 0.05 #define NUM\_SUCCESS\_NEEDED 1

int num\_success = 0; int num\_finished = 0;

void simulate1() {

if ((float)rand() / RAND\_MAX < SUCCESS\_PROBABILITY) {

num\_success++;

kill(getppid(), SIGUSR1);

} else {

kill(getppid(), SIGUSR2); }}

void simulate2() {

sleep(1);

}

void handle\_sigusr1(int sig) {

printf("Worker %d found relevant data.\n", getpid());

}

void handle\_sigusr2(int sig) {

printf("Worker %d did not find relevant data.\n", getpid());

}

void handle\_sigint(int sig) {

printf("Inefficient algorithm!\n");

// Terminate all child processes.

for (int i = 0; i < NUM\_WORKERS; i++) {

kill(i+1, SIGTERM);

}

exit(0);

}

int main() {

srand(time(NULL));

// Register signal handlers.

signal(SIGUSR1, handle\_sigusr1);

signal(SIGUSR2, handle\_sigusr2);

signal(SIGINT, handle\_sigint);

// Fork worker processes.

for (int i = 0; i < NUM\_WORKERS; i++) {

pid\_t pid = fork();

if (pid == 0) {

// Child process.

simulate1();

pause();

simulate2();

exit(0);

} else if (pid < 0) {

// Error forking process.

perror("fork");

exit(1);

}

}

// Wait for notifications from worker processes.

while (num\_finished < NUM\_WORKERS) {

pause();

num\_finished++;

if (num\_success >= NUM\_SUCCESS\_NEEDED && num\_finished >= NUM\_WORKERS/2) {

// Send SIGUSR1 signal to all worker processes.

for (int i = 0; i < NUM\_WORKERS; i++) {

kill(i+1, SIGUSR1);

}

break;

}

if (num\_finished == NUM\_WORKERS) {

printf("Inefficient algorithm!\n");

// Terminate all child processes.

for (int i = 0; i < NUM\_WORKERS; i++) {

kill(i+1, SIGTERM); }

break; }

}

for (int i = 0; i < NUM\_WORKERS; i++) {

wait(NULL); }

return 0;}

PL2 EX13

#define MAQUINAS 4#define PIECES 1000#define M1\_TIME 5#define M2\_TIME 5 #define M3\_TIME 10#define M4\_TIME 100

int main(){

pid\_t pids[MAQUINAS];

int fd[MAQUINAS][2];

int totalPecas = 0;

int pecas = 0;

int pecasRecebidas = 0;

for(int i = 0; i < MAQUINAS; i++) {

if (pipe(fd[i]) == -1){

perror("Pipe Falhou!");

exit(-1);

}

}

for (int i=0;i < MAQUINAS; i++){

pids[i] = fork();

if(pids[i] == -1){

perror("Fork Falhou");

return 2;

}

if (pids[i]==0){

close(fd[i][0]);

if (i>0){

close(fd[i-1][1]);

}

while(totalPecas < PIECES) {

if (i==0){

if (M1\_TIME == pecas){

totalPecas += pecas;

write(fd[i][1], &pecas, sizeof(pecas));

printf("M1 enviou %d peças para M2, tendo já enviado um total de %d peças. \n", pecas,totalPecas);

pecas = 0;

}

pecas++;

} else {

if (i==1){

read(fd[i-1][0], &pecasRecebidas, sizeof(pecas));

pecas += pecasRecebidas;

if (M2\_TIME == pecas){

totalPecas += pecas;

write(fd[i][1], &pecas, sizeof(pecas));

printf("M2 enviou %d peças para M3, tendo já enviado um total de %d peças.\n", pecas,totalPecas);

pecas = 0;

}

}

if (i==2){

read(fd[i-1][0], &pecasRecebidas, sizeof(pecas));

pecas += pecasRecebidas;

if (M3\_TIME == pecas){

totalPecas += pecas;

write(fd[i][1], &pecas, sizeof(pecas));

printf("M3 enviou %d peças para M4, tendo já enviado um total de %d peças.\n", pecas,totalPecas);

pecas = 0;

}

}

if (i==3){

read(fd[i-1][0], &pecasRecebidas, sizeof(pecas));

pecas += pecasRecebidas;

if (M4\_TIME == pecas){

totalPecas += pecas;

write(fd[i][1], &pecas, sizeof(pecas));

printf("M4 enviou %d peças para A1, tendo já enviado um total de %d peças.\n", pecas,totalPecas);

pecas = 0;

}

}

}

}

close(fd[i][1]);

exit(0);

}

}

close(fd[3][1]);

while(pecas < PIECES){

read(fd[3][0], &totalPecas, sizeof(totalPecas));

pecas += totalPecas;

}

close(fd[3][0]);

printf("Operação concluída com sucesso!\n");

return 0;}

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra, branco

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, informação

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

PL4 EX14

#include <semaphore.h> #include <sys/types.h> #include <unistd.h>

#include <stdio.h> #include <sys/syscall.h> #include <stdlib.h>

#include <string.h> #include <signal.h> #include <time.h>

#include <sys/wait.h> #include <ctype.h> #include <fcntl.h>

#include <sys/stat.h> #include <sys/mman.h>

#define SHARED\_NAME "/shared\_memory"

#define QUANTITY 10

#define pA 3

#define pB 2

typedef struct {

  int n[QUANTITY];

} SharedMemory;

int main(void){

    int random\_n[QUANTITY];

    int fd, size = sizeof(SharedMemory);

    SharedMemory \*shm;

    pid\_t pid;

    srand(time(NULL));

    //Creates and opens a shared memory area

    fd = shm\_open(SHARED\_NAME, O\_CREAT|O\_EXCL|O\_RDWR, S\_IRUSR|S\_IWUSR);

    if(fd < 0){

        perror("shm\_open");

        return 0;

    }

    //Defines the size

    if(ftruncate(fd, size) < 0){

        perror("ftruncate");

        return 0;

    }

    //Maps the shared area in the process address space

    shm = (SharedMemory\*)mmap(NULL, size, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0);

    sem\_t \*prA,\*prB,\*write;

    if ((prA = sem\_open("/prA", O\_CREAT|O\_EXCL, 0644, 1)) == SEM\_FAILED || (prB = sem\_open("/prB", O\_CREAT|O\_EXCL, 0644, 0)) == SEM\_FAILED || (write = sem\_open("/write", O\_CREAT|O\_EXCL, 0644, 1)) == SEM\_FAILED){

        perror("Error in sem\_open()");

        exit(1);

    }

    for (int i = 0; i < pA; i++) {

        pid = fork();

        if (pid == 0) {

            for(int j = 0; j < QUANTITY;j++){

                random\_n[j] = rand() % QUANTITY + 1;

                shm->n[j] = random\_n[j];

                printf("Process A produced the nber %d: %d\n",j+1,shm->n[j]);

            }

            sleep(5);

            sem\_wait(prA);

            sem\_wait(write);

            printf("\n");

            sem\_post(prA);

            sem\_post(write);

            sem\_post(prB);

            //Disconnects the shared memory area from the process address space

            munmap(shm, size);

            //Closes the file descritor

            close(fd);

            return 0;

        }

    }

    for(int i = 0; i < pB; i++){

        pid = fork();

        if(pid == 0){

            for(int j = 0; j < QUANTITY;j++){

                random\_n[j] = rand() % QUANTITY + 1;

                shm->n[j] = random\_n[j];

                printf("Process B produced the nber %d: %d\n",j+1,shm->n[j]);

            }

            sleep(6);

            sem\_wait(prB);

            sem\_wait(write);

            printf("\n");

            sem\_post(prB);

            sem\_post(write);

            sem\_post(prA);

//Disconnects the shared memory area from the process address space

            munmap(shm, size);

            //Closes the file descritor

            close(fd);

            return 0;        }    }

    //wait for all processes to finish

    for(int i = 0; i < (pA + pB);i++){

        wait(NULL);    }

    //unlink all the semaphores

    if (sem\_unlink("/prA") < 0 || sem\_unlink("/prB") < 0 || sem\_unlink("/write") < 0){

        perror("Error in sem\_unlink()");

        exit(1);

    }

    munmap(shm, size);

    close(fd);

    shm\_unlink(SHARED\_NAME);

    return 0;

}

PL4 EX16

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

PL5 EX4

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define MATRIX\_SIZE 8

int matrix1[MATRIX\_SIZE][MATRIX\_SIZE];

int matrix2[MATRIX\_SIZE][MATRIX\_SIZE];

int result[MATRIX\_SIZE][MATRIX\_SIZE];

// Função executada pela primeira thread para preencher a primeira matriz

void\* fillMatrix1(void\* arg) {

    int value = \*(int\*)arg;

    for (int i = 0; i < MATRIX\_SIZE; i++) {

        for (int j = 0; j < MATRIX\_SIZE; j++) {

            matrix1[i][j] = value;

        }

    }

    return NULL;

}

// Função executada pela segunda thread para preencher a segunda matriz

void\* fillMatrix2(void\* arg) {

    int value = \*(int\*)arg;

    for (int i = 0; i < MATRIX\_SIZE; i++) {

        for (int j = 0; j < MATRIX\_SIZE; j++) {

            matrix2[i][j] = value;

        }

    }

    return NULL;

}

// Função executada pelas outras threads para realizar a multiplicação das matrizes

void\* multiplyMatrices(void\* arg) {

    int row = \*(int\*)arg;

    for (int i = row; i < row + 1; i++) {

        for (int j = 0; j < MATRIX\_SIZE; j++) {

            result[i][j] = 0;

            for (int k = 0; k < MATRIX\_SIZE; k++) {

                result[i][j] += matrix1[i][k] \* matrix2[k][j];

            }

        }

    }

    return NULL;

}

int main() {

    pthread\_t fill\_thread1, fill\_thread2;

    pthread\_t multiply\_threads[MATRIX\_SIZE];

    // Cria as threads para preencher as matrizes

    int value1 = 2;

    int value2 = 3;

    pthread\_create(&fill\_thread1, NULL, fillMatrix1, &value1);

    pthread\_create(&fill\_thread2, NULL, fillMatrix2, &value2);

    // Aguarda que as outras threads terminem o preenchimento

    pthread\_join(fill\_thread1, NULL);

    pthread\_join(fill\_thread2, NULL);

    // Cria as threads para realizar a multiplicação das matrizes

    for (int i = 0; i < MATRIX\_SIZE; i++) {

        int\* row = malloc(sizeof(int));

        \*row = i;

        pthread\_create(&multiply\_threads[i], NULL, multiplyMatrices, row);

    }

    // Aguarda que as outras threads terminem o processo de multiplicação

    for (int i = 0; i < MATRIX\_SIZE; i++) {

        pthread\_join(multiply\_threads[i], NULL);

    }

    // Imprime a matriz resultante

    printf("Matriz resultante:\n");

    for (int i = 0; i < MATRIX\_SIZE; i++) {

        for (int j = 0; j < MATRIX\_SIZE; j++) {

            printf("%d ", result[i][j]);

        }

        printf("\n");    }

    return 0;}

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, círculo, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

PL5 EX10

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define NUM\_THREADS 6

#define NUM\_HYPERMARKETS 3

#define NUM\_PRODUCTS 5

#define ARRAY\_SIZE 10000

// Estrutura para armazenar os dados de cada elemento do array Vec

typedef struct {

    int hypermarket\_id;

    int product\_id;

    double price;

} Data;

Data Vec[ARRAY\_SIZE];  // Array para armazenar os dados que foram retirados

double lowest\_cost = \_\_DBL\_MAX\_\_;  // Variável global para armazenar o menor custo

int lowest\_hypermarket\_id;  // Variável global para armazenar o id do hipermercado com menor custo

pthread\_mutex\_t mutex[NUM\_HYPERMARKETS];  // Array de mutex para garantir exclusão mútua ao acessar os arrays específicos do hipermercado

double calculate\_average\_price(Data\* data, int size) {

    double sum = 0.0;

    for (int i = 0; i < size; i++) {

        sum += data[i].price;

    }

    return sum / size;

}

void \*filter\_data(void \*arg) {

    int hypermarket\_id = \*(int \*)arg;

    Data filtered\_data[NUM\_PRODUCTS];

    int filtered\_data\_count = 0;

    // Filtrar os dados com base no id do hipermercado

    for (int i = 0; i < ARRAY\_SIZE; i++) {

        if (Vec[i].hypermarket\_id == hypermarket\_id) {

            filtered\_data[filtered\_data\_count++] = Vec[i];

        }

    }

    // Bloquear o acesso ao array específico do hipermercado usando o mutex correspondente

    pthread\_mutex\_lock(&mutex[hypermarket\_id - 1]);

    // Desbloquear o mutex

    pthread\_mutex\_unlock(&mutex[hypermarket\_id - 1]);

    // Sinalizar o término do estágio de filtragem

    pthread\_exit(NULL);

}

void \*compute\_cost(void \*arg) {

    int hypermarket\_id = \*(int \*)arg;

    Data\* data;

    int data\_size;

    // Calcular o preço médio de cada produto

    double average\_prices[NUM\_PRODUCTS];

    // Calcular a soma dos preços médios dos produtos

    double sum = 0.0;

    for (int i = 0; i < NUM\_PRODUCTS; i++) {

        sum += average\_prices[i];

    }

    // término do estágio de cálculo

    pthread\_exit(NULL);

}

int main() {

    // Inicializar o array Vec com os dados retirados

    pthread\_t threads[NUM\_THREADS];

    int hypermarket\_ids[NUM\_HYPERMARKETS] = {1, 2, 3};

    // Inicializar os mutex

    for (int i = 0; i < NUM\_HYPERMARKETS; i++) {

        pthread\_mutex\_init(&mutex[i], NULL);

    }

    // Criar as threads que irão filtrar os id

    for (int i = 0; i < NUM\_HYPERMARKETS; i++) {

        pthread\_create(&threads[i], NULL, filter\_data, &hypermarket\_ids[i]);

    }

    // Aguardar o término das threads de filtragem

    for (int i = 0; i < NUM\_HYPERMARKETS; i++) {

        pthread\_join(threads[i], NULL);

    }

    // Criar as threads de cálculo

    for (int i = NUM\_HYPERMARKETS; i < NUM\_THREADS; i++) {

        pthread\_create(&threads[i], NULL, compute\_cost, &hypermarket\_ids[i - NUM\_HYPERMARKETS]);

    }

    // Aguardar o término das threads de cálculo

    for (int i = NUM\_HYPERMARKETS; i < NUM\_THREADS; i++) {

        pthread\_join(threads[i], NULL);

    }

    // Imprimir o menor custo e o id do hipermercado

    printf("Menor custo: %f\n", lowest\_cost);

    printf("Hipermercado com menor custo: %d\n", lowest\_hypermarket\_id);

    // Destruir os mutex

    for (int i = 0; i < NUM\_HYPERMARKETS; i++) {

        pthread\_mutex\_destroy(&mutex[i]);    }

    return 0;}