**[Practice Exam 1 :](#Ex1)**

/\*Scrieti un program C care genereaza N threaduri (N este dat la linia de comanda. Fiecare thread primeste un index I ca argument. Fiercare thread genereaza un numar aleator A intre 1 si 1000 si il va insera pe pozitia I intr-un sir accesibil tututor thread-urilor. Dupa ce toate thread-urile au inserat numarul A in sir, fiecare thread va genera un alt numar aleator B intre 1 si 1000. Fiecare thread va cauta in sir cel mai mic numar mai mare decat B, va scadea numarul B din el si va afisa valorile din sir dupa scadere, daca nu exista un astfel de numar threadul va afisa un messaj corespunzator si va afisa sirul. Fiecare thread va afisa index-ul propriu si numerele generate A si B. Folositi mecanisme eficiente de sincronizare.\*/

[**Practice Exam 2 :**](#Ex2)

/\* Scrieti un program C care creeaza 2 procese fiu. Procesul parinte genereza un numar aleator N intre 100 si 1000. Procesele fiu genereaza pe rand cate un numar intre 1 si 100 si il trimit procesului parinte. Procesul parinte citeste un numar de la primul proces fiu si, daca este mai mic decat N, scade numarul din N. Apoi procesul parinte citeste un numar de la al doilea proces fiu, si daca este mai mic decat noua valoare a lui N, scade numarul din aceasta. Acest proces se repeta pana cand unul dintre procesele fiu genereaza un numar care nu poate fi scazut din N, moment in care ambele procese fiu isi incheie executia. Procesul parinte va afisa numarul genereat N, numerele primite de la procesele fiu, alaturi de procesul de la care au fost primite si valorile lui N dupa fieacare scadere. \*/

**Practice Exam 1 :**

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <string.h>

typedef struct{

int idx;

int\* array;

pthread\_mutex\_t\* mtx;

pthread\_barrier\_t\* bar;

} data;

int n;

void initData(data\* d, int givenIdx, int\* givenArray, pthread\_mutex\_t\* givenMtx, pthread\_barrier\_t\* givenBar) {

d->idx = givenIdx;

d->array = givenArray;

d->mtx = givenMtx;

d->bar = givenBar;

}

void\* thread(void\* a) {

data d = \*((data\*)a);

int i;

int A = random() % 1001 + 1;

d.array[d.idx] = A;

printf("Waiting...\n");

pthread\_barrier\_wait(d.bar);

int B = random() % 1001 + 1;

printf("\nThread %d - generated %d and %d\n", d.idx, A, B);

pthread\_mutex\_lock(d.mtx);

printf("\nThread %d :", d.idx);

int ok = 1;

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

if (d.array[i] > B) {

d.array[i] -= B;

ok = 0;

}

printf("%d, ", d.array[i]);

}

if (ok == 1)

printf("\nNo satisfying value was found\n");

else

printf("\n");

pthread\_mutex\_unlock(d.mtx);

return NULL;

}

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

if (argc != 2) {

perror("Please provide 1 integer argument");

exit(1);

}

n = atoi(argv[1]);

int i;

pthread\_mutex\_t\* mtx = malloc(sizeof(pthread\_mutex\_t));

pthread\_mutex\_init(mtx, NULL);

pthread\_barrier\_t\* bar = malloc(sizeof(pthread\_barrier\_t));

pthread\_barrier\_init(bar, NULL, n);

data\* arg = malloc(sizeof(data) \* n);

int\* array = malloc(sizeof(int) \* n);

memset(array, 0, sizeof(int) \* (n + 1));

pthread\_t\* T = malloc(sizeof(pthread\_t) \* n);

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

initData(&arg[i], i, array, mtx, bar);

pthread\_create(&T[i], NULL, thread, (void\*)&arg[i]);

}

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

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

//pthread\_barrier\_destory(&bar);

free(mtx);

free(bar);

free(array);

free(arg);

free(T);

return 0;

}

**Practice Exam 2 :**

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <pthread.h>

#include <fcntl.h>

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

if (argc != 1) {

perror("No argument shoul be provided");

exit(1);

}

int a2p[2], p2a[2], b2p[2], p2b[2];

if (0 > pipe(a2p)) {

perror("Error on pipe a2p");

exit(1);

}

if (0 > pipe(p2a)) {

perror("Error on pipe p2a");

exit(1);

}

if (0 > pipe(b2p)) {

perror("Error on pipe b2p");

exit(1);

}

if (0 > pipe(p2b)) {

perror("Error on pipe p2b");

exit(1);

}

int fa = fork();

if (0 > fa) {

perror("Error on creating child process a");

exit(1);

}

else if (fa == 0) {

close(a2p[0]);

close(p2a[1]);

close(b2p[0]);

close(b2p[1]);

close(p2b[0]);

close(p2b[1]);

while (1) {

srandom(getpid());

int a = random() % 100 + 1;

if (0 > write(a2p[1], &a, sizeof(int))) {

perror("Error on writing to pipe a2p");

exit(1);

}

if (0 > read(p2a[0], &a, sizeof(int))) {

perror("Error on reading from pipe p2a");

exit(1);

}

if (a == -1)

break;

}

close(a2p[1]);

close(p2a[0]);

exit(0);

}

int fb = fork();

if (0 > fb) {

perror("Error on creating child process b");

exit(1);

}

else if (fb == 0) {

close(b2p[0]);

close(p2b[1]);

close(a2p[0]);

close(a2p[1]);

close(p2a[0]);

close(p2a[1]);

while (1) {

int b = random() % 100 + 1;

if (0 > write(b2p[1], &b, sizeof(int))) {

perror("Error on writing to pipe b2p");

exit(1);

}

if (0 > read(p2b[0], &b, sizeof(int))) {

perror("Error on reading from pipe p2b");

exit(1);

}

if (b == -1)

break;

}

close(b2p[1]);

close(p2b[0]);

exit(0);

}

close(a2p[1]);

close(p2a[0]);

close(b2p[1]);

close(p2b[0]);

srandom(getpid());

int n = random() % 901 + 100;

printf("Parent - generated %d\n", n);

int a, b, temp;

int stop = 0;

while (stop == 0) {

if (0 > read(a2p[0], &a, sizeof(int))) {

perror("Error on reading data from pipe a2p");

exit(1);

}

printf("Parent - received from A %d\n", a);

if (a < n) {

temp = n;

n -= a;

printf("%d -> %d\n", temp, n);

}

else

stop = -1;

if (0 > write(p2b[1], &stop, sizeof(int))) {

perror("Error on writing data to pipe d2b");

exit(1);

}

if (0 > write(p2a[1], &stop, sizeof(int))) {

perror("Error on writing data to pipe d2b");

exit(1);

}

if (0 > read(b2p[0], &b, sizeof(int))) {

perror("Error on reading data from pipe b2p");

exit(1);

}

printf("Parent - received from B %d\n", b);

if (b < n) {

temp = n;

n -= b;

printf("%d -> %d\n", temp, n);

}

else

stop = -1;

if (0 > write(p2b[1], &stop, sizeof(int))) {

perror("Error on writing data to pipe d2b");

exit(1);

}

if (0 > write(p2a[1], &stop, sizeof(int))) {

perror("Error on writing data to pipe d2b");

exit(1);

}

}

wait(0);

wait(0);

close(a2p[0]);

close(p2a[1]);

close(b2p[0]);

close(p2b[1]);

printf("Done!\n");

return 0;

}