**ASSIGNMENT1**

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include<iostream>

using namespace std;

int\* n;

int main(){

// create shared memory for 14 int variables

// first 2 ints for flags abReady, cReady

// next 12 for keeping the 3 2x2 matrices

n = (int\*)mmap(NULL, sizeof(int)\*14,

PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

int\* abReady = n;

int\* cReady = n+1;

int\* a = cReady+1;

int\* b = a+4;

int\* c = b+4;

int pid = fork();

if(pid==0){ //it's the child

\*abReady = 0;

\*cReady = 0;

cout<<"[CHILD]"<<endl;

cout<<"Enter matrix A's elements (4 inputs): ";

cin>>a[0]>>a[1]>>a[2]>>a[3];

cout<<"Enter matrix B's elements (4 inputs): ";

cin>>b[0]>>b[1]>>b[2]>>b[3];

cout<<"Matrix A:\n"<<a[0]<<" "<<a[1]<<"\n"<<a[2]<<" "<<a[3]<<"\n";

cout<<"Matrix B:\n"<<b[0]<<" "<<b[1]<<"\n"<<b[2]<<" "<<b[3]<<"\n";

\*abReady = 1;

while(!\*cReady) sleep(5);

cout<<"[CHILD]"<<endl;

cout<<"Matrix C:\n"<<c[0]<<" "<<c[1]<<"\n"<<c[2]<<" "<<c[3]<<"\n";

exit(0);

}else{ // its the parent

while(!\*abReady) sleep(5);

cout<<"[PARENT] (calculating c = a \* b)"<<endl;

c[0] = a[0]\*b[0] + a[1]\*b[2];

c[1] = a[0]\*b[1] + a[1]\*b[3];

c[2] = a[2]\*b[0] + a[3]\*b[2];

c[3] = a[2]\*b[1] + a[3]\*b[3];

\*cReady = 1;

wait(0);

}

exit(0);

}

**ASSIGNMENT2**

#include<stdio.h>

#include<unistd.h>

#include<cstdlib>

#include<cstring>

using namespace std;

int\* n;

int main(){

int pipeA[2], a;

int pipeB[2], b;

a = pipe(pipeA);

b = pipe(pipeB);

if(a == -1) printf("pipe A failed!\n");

else printf("pipe A made!\n");

if(b == -1) printf("pipe B failed!\n");

else printf("pipe B made!\n");

char \*line0 = "hello! this is line 0!";

char \*line1 = "hello again. this is the other line! it's line 1.";

int pA = fork();

if(pA==0){ //it's child A

close(pipeA[0]);

for(int i=0; i<strlen(line0); i++){

if(line0[i]=='!') break;

write(pipeA[1], line0+i, 1);

}

close(pipeA[1]);

exit(0);

}

else{ // its the parent

int pB = fork();

if(pB==0){ // it's child B

close(pipeB[0]);

for(int i=0; i<strlen(line1); i++){

if(line1[i]=='!') break;

write(pipeB[1], line1+i, 1);

}

close(pipeB[1]);

exit(0);

}else{ // its the parent

close(pipeA[1]);

close(pipeB[1]);

char buf;

printf("[parent] reading pipe A\n");

while(read(pipeA[0], &buf, 1) > 0) printf("%c",buf);

close(pipeA[0]);

printf("\n");

printf("[parent] reading pipe B\n");

while(read(pipeB[0], &buf, 1) > 0) printf("%c",buf);

close(pipeB[0]);

}

}

exit(0);

}

**ASSIGNMENT3**

package os.ass.pkg03;

import java.util.concurrent.atomic.AtomicInteger;

public class OSASS03 {

public static void main(String[] args) {

AtomicInteger counter = new AtomicInteger(0);

int n = Integer.parseInt(args[0]);

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

Thread t = new Thread(new Runnable() {

@Override

public void run() {

for(int i=0; i<10000; i++){

counter.addAndGet(1);

}

System.out.println("Counter = "+String.valueOf(counter.get()));

}

});

t.start();

}

}

}

// PS > java -jar .\OS\_ASS\_03.jar 2

// Counter = 20000

// Counter = 20000

// PS > java -jar .\OS\_ASS\_03.jar 4

// Counter = 40000

// Counter = 40000

// Counter = 40000

// Counter = 40000

// PS > java -jar .\OS\_ASS\_03.jar 8

// Counter = 38013

// Counter = 40259

// Counter = 40259

// Counter = 37988

// Counter = 69894

// Counter = 72657

// Counter = 74895

// Counter = 80000

**ASSIGNMENT4**

#include<iostream>

#include<cstdlib>

#include<pthread.h>

#include<semaphore.h>

using namespace std;

#define NUM\_THREADS 10

sem\_t rw\_mutex;

sem\_t mutex;

int readers = 0;

int data = 25;

void \*write(void \*t){

sem\_wait(&rw\_mutex);

cout<<"[WRITER t="<<t<<"] writing data = "<<data;

for(int i=0; i<rand()%NUM\_THREADS; i++) cout<<">"<<++data;

cout<<" end!"<<endl;

sem\_post(&rw\_mutex);

}

void \*read(void \*t){

sem\_wait(&mutex);

readers++;

if(readers == 1) sem\_wait(&rw\_mutex);

sem\_post(&mutex);

cout<<"[READER t="<<t<<"] reading data = "<<data;

for(int i=0; i<rand()%NUM\_THREADS; i++) cout<<"."<<data;

cout<<" end!"<<endl;

sem\_wait(&mutex);

readers--;

if(readers == 0) sem\_post(&rw\_mutex);

sem\_post(&mutex);

}

int main(){

if(sem\_init(&rw\_mutex, 0, 1) == -1) cout<<"SEM rw\_mutex error!"<<endl;

if(sem\_init(&mutex, 0, 1) == -1) cout<<"SEM mutex error!"<<endl;

int rc; long t;

const int WRITER = rand()%NUM\_THREADS;

cout<<"Write ID: "<<WRITER<<endl;

pthread\_t threads[NUM\_THREADS];

for(t=0;t<NUM\_THREADS;t++){

cout<<"creating thread "<<t<<endl;

if(t==WRITER) rc = pthread\_create(&threads[t], NULL, write, (void \*)t);

else rc = pthread\_create(&threads[t], NULL, read, (void \*)t);

if (rc){

cout<<"ERROR; return code from pthread\_create() is "<<rc<<endl;

exit(-1); }

}

pthread\_exit(NULL);

return 0; }

**ASSIGNMENT5**

#include<iostream>

#include<cstdlib>

#include<pthread.h>

#include<semaphore.h>

using namespace std;

#define NUM\_BUFFER 10

#define NUM\_THREAD 2

#define LIMIT 10

sem\_t mutex;

sem\_t empty;

sem\_t full;

int\* buffer = new int[NUM\_BUFFER];

int offset = 0;

int production\_count = LIMIT;

int consumption\_count = 0;

void \*produce(void \*t){

do{

int data = rand()%100;

cout<<"[PRODUCER t="<<t<<"] produced data = "<<data<<endl;

production\_count--;

sem\_wait(&empty);

sem\_wait(&mutex);

cout<<"[PRODUCER t="<<t<<"] adding data = "<<data<<endl;

buffer[offset++] = data;

sem\_post(&mutex);

sem\_post(&full);

}while(production\_count>0);

}

void \*consume(void \*t){

do{

sem\_wait(&full);

sem\_wait(&mutex);

int data = buffer[offset--];

cout<<"[CONSUMER t="<<t<<"] removed data = "<<data<<endl;

sem\_post(&mutex);

sem\_post(&empty);

cout<<"[CONSUMER t="<<t<<"] consuming data = "<<data<<endl;

consumption\_count++;

}while(consumption\_count<LIMIT);

}

int main(){

if(sem\_init(&mutex, 0, 1) == -1) cout<<"SEM mutex error!"<<endl;

if(sem\_init(&empty, 0, NUM\_BUFFER) == -1) cout<<"SEM empty error!"<<endl;

if(sem\_init(&full, 0, 0) == -1) cout<<"SEM full error!"<<endl;

int rc; long t;

pthread\_t threads[NUM\_THREAD];

for(t=0;t<NUM\_THREAD;t++){

cout<<"creating thread "<<t<<endl;

if(t%2==0) rc = pthread\_create(&threads[t], NULL, produce, (void \*)t);

else rc = pthread\_create(&threads[t], NULL, consume, (void \*)t);

if (rc){

cout<<"ERROR; return code from pthread\_create() is "<<rc<<endl;

exit(-1);

}

}

pthread\_exit(NULL);

return 0;

}