OS Lab Assignment 4

IMPLEMENTATION OF CASSICAL IPC PROBEMS USING PTHREADS

1. Dining Philosopher Thread

// Dining\_Philospher\_Thread

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

pthread\_t philosopher[5];

pthread\_mutex\_t chopstick[5];

void \*func(int n);

int main(){

  int i,k;

  void \*msg;

  for(i=1;i<=5;i++){

    k=pthread\_mutex\_init(&chopstick[i],NULL);

    if(k==-1){

      printf("\n Mutex initialization failed");

      exit(1);

    }

  }

  for(i=1;i<=5;i++){

    k=pthread\_create(&philosopher[i],NULL,(void \*)func, (int \*)(intptr\_t)i);

    if(k!=0){

      printf("\n Thread creation error \n");

      exit(1);

    }

  }

  for(i=1;i<=5;i++){

    k=pthread\_join(philosopher[i],&msg);

    if(k!=0){

      printf("\n Thread join failed \n");

      exit(1);

    }

  }

  for(i=1;i<=5;i++){

    k=pthread\_mutex\_destroy(&chopstick[i]);

    if(k!=0){

      printf("\n Mutex Destroyed \n");

      exit(1);

    }

  }

  return 0;

}

void \*func(int n){

  printf("\nPhilosopher %d is thinking ",n);

  pthread\_mutex\_lock(&chopstick[n]);//when philosopher 5 is eating he takes fork 1 and fork 5 pthread\_mutex\_lock(&chopstick[(n+1)%5]);

  printf("\nPhilosopher %d is eating ",n);

  sleep(3);

  pthread\_mutex\_unlock(&chopstick[n]);

  pthread\_mutex\_unlock(&chopstick[(n+1)%5]);

  printf("\nPhilosopher %d Finished eating ",n);

}

1. Producer Consumer

// Producer\_Consumer

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

sem\_t empty, full, mutex;

void \*producer (void \*);

void \*consumer(void \*);

int buffer[10];

int ID[10];

int in = 0 ; int out = 0;

int BUFFER\_SIZE = 10;

int nextProduced = 0;

int main(){

  int i;

  pthread\_t TID[10];

  sem\_init(&empty, 0, 10);

  sem\_init(&full, 0, 0);

  sem\_init(&mutex, 0, 1);

  for(i = 0; i < 10; i++){

  ID[i] = i;

  buffer[i] = -1;

  }

  pthread\_create(&TID[0], NULL, producer, (void \*) &ID[0]) ; printf("Producer ID = %d created!\n", 0);

  pthread\_create(&TID[1], NULL, consumer, (void \*) &ID[1]) ; printf("Consumer ID = %d created!\n", 1);

  pthread\_create(&TID[2], NULL, producer, (void \*) &ID[2]) ; printf("Producer ID = %d created!\n", 2);

  pthread\_create(&TID[3], NULL, consumer, (void \*) &ID[3]) ; printf("Consumer ID = %d created!\n", 3);

  for(i = 0; i < 10 ; i++)pthread\_join(TID[i], NULL);

}

void \*producer(void \*Boo){

  int \*ptr;

  int ID;

  ptr = (int \*) Boo;

  ID = \*ptr;

  while (1){

  nextProduced++; //Producing Integers

  /\* Check to see if Overwriting unread slot \*/

  sem\_wait(&empty);

  sem\_wait(&mutex);

  if (buffer[in] != -1){

    printf("Synchronization Error: Producer %d Just overwrote %d from Slot %d \n ", ID, buffer[in], in);

    exit(0);

  }

  buffer[in] = nextProduced;

  printf("Producer %d. Put %d in slot %d\n", ID, nextProduced, in);

  in = (in + 1) % BUFFER\_SIZE;

  printf("incremented in!\n");

  sem\_post(&mutex);

  sem\_post(&full);

  }

}

void \*consumer (void \*Boo){

  static int nextConsumed = 0;

  int \*ptr;

  int ID;

  ptr = (int \*) Boo;

  ID = \*ptr;

  while (1){

    sem\_wait(&full);

    sem\_wait(&mutex);

    nextConsumed = buffer[out];

    if (nextConsumed == -1){

      printf("Synch Error: Consumer %d Just Read from empty slot %d\n", ID, out);

      exit(0);

    }

    printf("Consumer %d Just consumed item %d from slot %d\n", ID, nextConsumed, out);

    buffer[out] = -1;

    out = (out + 1) % BUFFER\_SIZE;

    sem\_post(&mutex);

    sem\_post(&empty);

  }

}

1. Reader Writer

// Reader\_Writer

#include<semaphore.h>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

sem\_t x,y;

pthread\_t tid;

pthread\_t writerthreads[100],readerthreads[100];

int readercount = 0;

void \*reader(void\* param){

  sem\_wait(&x);

  readercount++;

  if(readercount==1)sem\_wait(&y);

  sem\_post(&x);

  printf("%d reader is inside\n",readercount);

  usleep(3);

  sem\_wait(&x);

  readercount--;

  if(readercount==0)sem\_post(&y);

  sem\_post(&x);

  printf("%d Reader is leaving\n",readercount+1);

  return NULL;

}

void \*writer(void\* param){

  printf("Writer is trying to enter\n");

  sem\_wait(&y);

  printf("Writer has entered\n");

  sem\_post(&y);

  printf("Writer is leaving\n");

  return NULL;

}

int main(){

  int n2,i;

  printf("Enter the number of readers:");

  scanf("%d",&n2);

  printf("\n");

  int n1[n2];

  sem\_init(&x,0,1);

  sem\_init(&y,0,1);

  for(i=0;i<n2;i++){

    pthread\_create(&writerthreads[i],NULL,reader,NULL);

    pthread\_create(&readerthreads[i],NULL,writer,NULL);

  }

  for(i=0;i<n2;i++){

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

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

  }

}

1. Sleeping Barber

//Sleeping\_Barber

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define MAX 20

sem\_t chairs\_mutex;

sem\_t sem\_client;

sem\_t sem\_barber;

void \*client(void \*param);

void \*barber(void \*param);

int num\_chairs;

int clientWait;

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

  pthread\_t barberid;

  pthread\_t clientids[MAX];

  printf("Main thread beginning\n");

  /\* 1. Get command line arguments \*/

  int runTime,clients,i;

  if (argc != 5){

    printf("Please enter 4 arguments: <Program run time> <Number of clients>\n");

    printf("<Number of chairs> <Client wait time>\n");

    exit(0);

  }

  runTime = atoi(argv[1]);

  clients = atoi(argv[2]);

  num\_chairs = atoi(argv[3]);

  clientWait = atoi(argv[4]);

  /\* 2. Initialize semaphores \*/

  sem\_init(&chairs\_mutex,0,1);

  sem\_init(&sem\_client,0,0);

  sem\_init(&sem\_barber,0,0);

  /\* 3. Create barber thread. \*/

  pthread\_create(&barberid, NULL, barber, NULL);

  printf("Creating barber thread with id %lu\n",barberid);

  /\* 4. Create client threads. \*/

  for (i = 0; i < clients; i++){

    pthread\_create(&clientids[i], NULL, client, NULL);

    printf("Creating client thread with id %lu\n",clientids[i]);

  }

  /\* 5. Sleep. \*/

  printf("Main thread sleeping for %d seconds\n", runTime);

  sleep(runTime);

  /\* 6. Exit. \*/

  printf("Main thread exiting\n");

  exit(0);

}

void \*barber(void \*param) {

  int worktime;

  while(1) {

    /\* wait for a client to become available (sem\_client) \*/

    sem\_wait(&sem\_client);

    /\* wait for mutex to access chair count (chair\_mutex) \*/

    sem\_wait(&chairs\_mutex);

    /\* increment number of chairs available \*/

    num\_chairs += 1;

    printf("Barber: Taking a client. Number of chairs available = %d\n",num\_chairs);

    /\* signal to client that barber is ready to cut their hair (sem\_barber) \*/

    sem\_post(&sem\_barber);

    /\* give up lock on chair count \*/

    sem\_post(&chairs\_mutex);

    /\* generate random number, worktime, from 1-4 seconds for length of haircut. \*/

    worktime = (rand() % 4) + 1;

    /\* cut hair for worktime seconds (really just call sleep()) \*/

    printf("Barber: Cutting hair for %d seconds\n", worktime);

    sleep(worktime);

  }

}

void \*client(void \*param){

  int waittime;

  while(1){

    /\* wait for mutex to access chair count (chair\_mutex) \*/

    sem\_wait(&chairs\_mutex);

    /\* if there are no chairs \*/

    if(num\_chairs <= 0){

      /\* free mutex lock on chair count \*/

      printf("Client: Thread %u leaving with no haircut\n", (unsigned int)pthread\_self());

      sem\_post(&chairs\_mutex);

    }

    /\* else if there are chairs \*/

    else{

      /\* decrement number of chairs available \*/

      num\_chairs -= 1;

      printf("Client: Thread %u Sitting to wait. Number of chairs left = %d\n",(unsigned int)pthread\_self(),num\_chairs);

      /\* signal that a customer is ready (sem\_client) \*/

      sem\_post(&sem\_client);

      /\* free mutex lock on chair count \*/

      sem\_post(&chairs\_mutex);

      /\* wait for barber (sem\_barber) \*/

      sem\_wait(&sem\_barber);

      /\* get haircut \*/

      printf("Client: Thread %u getting a haircut\n",(unsigned int)pthread\_self());

    }

    /\* generate random number, waittime, for length of wait until next haircut or next try.

    Max value from command line. \*/

    waittime = (rand() % clientWait) + 1;

    /\* sleep for waittime seconds \*/

    printf("Client: Thread %u waiting %d seconds before attempting next haircut\n", (unsigned int)pthread\_self(),waittime);

    sleep(waittime);

  }

}

IMPLEMENTATION OF FOUR CLASSICAL IPC PROBLEMS USING SYSTEM V IPC FACILITIES.

1. Dining Philosopher

// Dining Philospher

#include<stdio.h>

#include<fcntl.h>

#include<semaphore.h>

#include<sys/wait.h>

#include<pthread.h>

#include<stdlib.h>

#include<unistd.h>

sem\_t \*sem[20];

int n;

int reader(int val);

int main(){

  pid\_t cpid[5];

  char semname[5];

  int i,j=0;

  n = 5;

  for(i=0;i<n;i++){

    sprintf(semname,"%d",getpid()+i);

    sem[i]=sem\_open(semname,O\_CREAT|O\_EXCL,0666,1);

    if(sem[i]==SEM\_FAILED) perror("Unable to create semaphore");

  }

  for(i=0;i<n;i++){

    cpid[i]=fork();

    if(cpid[i]==0)break;

  }

  if(i==n){

    int status;

    for(i=0;i<n;i++) waitpid(cpid[i],&status,WUNTRACED);

    //waitpid is a function which waits for the child process to finish executing

    //after that control switches back to parent

    for(i=0;i<n;i++){

      sem\_close(sem[i]);

      sprintf(semname,"%d ",getpid()+i);

      sem\_unlink(semname);

    }

  }

  else reader(i);

}

int reader(int val){

  printf("%d Thinking\n",val+1);

  while(1){

    sem\_wait(sem[val%n]);

    if(!sem\_trywait(sem[(val+1)%n]))break;

    else sem\_post(sem[val%n]);

  }

  printf("%d Eating\n",val+1);

  sleep(2);

  sem\_post(sem[val%n]);

  sem\_post(sem[(val+1)%n]);

  printf("%d Finished Eating\n",val+1);

}

1. Producer Consumer

//Producer\_Consumer

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

/\*

This program provides a possible solution for producer-consumer problem using

mutex and semaphore.

I have used 5 producers and 5 consumers to demonstrate the solution. You can always

play with these values.

\*/

#define MaxItems 5 // Maximum items a producer can produce or a consumer can consume

#define BufferSize 5 // Size of the buffer

sem\_t empty;

sem\_t full;

pthread\_mutex\_t mutex;

int in = 0;

int out = 0;

int buffer[BufferSize];

void \*producer(void \*pno){

  int item;

  for(int i = 0; i < MaxItems; i++){

    item = rand(); // Produce an random item

    sem\_wait(&empty);

    pthread\_mutex\_lock(&mutex);

    buffer[in] = item;

    printf("Producer %d: Insert Item %d at %d\n", \*((int \*)pno),buffer[in],in);

    in = (in+1)%BufferSize;

    pthread\_mutex\_unlock(&mutex);

    sem\_post(&full);

  }

}

void \*consumer(void \*cno){

  for(int i = 0; i < MaxItems; i++){

    sem\_wait(&full);

    pthread\_mutex\_lock(&mutex);

    int item = buffer[out];

    printf("Consumer %d: Remove Item %d from %d\n",\*((int \*)cno),item, out);

    out = (out+1)%BufferSize;

    pthread\_mutex\_unlock(&mutex);

    sem\_post(&empty);

  }

}

int main(){

  pthread\_t pro[5],con[5];

  pthread\_mutex\_init(&mutex, NULL);

  sem\_init(&empty,0,BufferSize);

  sem\_init(&full,0,0);

  int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

  for(int i = 0; i < 5; i++) pthread\_create(&pro[i], NULL, (void \*)producer, (void \*)&a[i]);

  for(int i = 0; i < 5; i++) pthread\_create(&con[i], NULL, (void \*)consumer, (void \*)&a[i]);

  for(int i = 0; i < 5; i++) pthread\_join(pro[i], NULL);

  for(int i = 0; i < 5; i++) pthread\_join(con[i], NULL);

  pthread\_mutex\_destroy(&mutex);

  sem\_destroy(&empty);

  sem\_destroy(&full);

  return 0;

}

1. Read Write

//Read\_Write

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t mutex,writeblock;

int data = 0,rcount = 0;

void \*reader(void \*arg){

  int f;

  f = ((intptr\_t)arg);

  sem\_wait(&mutex);

  rcount = rcount + 1;

  if(rcount==1) sem\_wait(&writeblock);

  sem\_post(&mutex);

  printf("Data read by the reader%d is %d\n",f,data); sleep(1);

  sem\_wait(&mutex);

  rcount = rcount - 1;

  if(rcount==0) sem\_post(&writeblock);

  sem\_post(&mutex);

}

void \*writer(void \*arg){

  int f;

  f = ((intptr\_t) arg);

  sem\_wait(&writeblock);

  data++;

  printf("Data writen by the writer%d is %d\n",f,data); sleep(1);

  sem\_post(&writeblock);

}

int main(){

  int i,b;

  pthread\_t rtid[5],wtid[5];

  sem\_init(&mutex,0,1);

  sem\_init(&writeblock,0,1);

  for(i=0;i<=2;i++)

    pthread\_create(&wtid[i],NULL,writer,(void \*)(intptr\_t)i); pthread\_create(&rtid[i],NULL,reader,(void \*)(intptr\_t)i);

  for(i=0;i<=2;i++){

    pthread\_join(wtid[i],NULL);

    pthread\_join(rtid[i],NULL);

  }

  return 0;

}

1. Sleeping Barber

//Sleeping\_Barber

#define \_REENTRANT

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define MAX\_CUSTOMERS 25 // The maximum number of customer threads.

// Function prototypes...

void \*customer(void \*num);

void \*barber(void \*);

void randwait(int secs);

// Define the semaphores.

// waitingRoom Limits the # of customers allowed

// to enter the waiting room at one time.

sem\_t waitingRoom;

// barberChair ensures mutually exclusive access to

// the barber chair.

sem\_t barberChair;

// barberPillow is used to allow the barber to sleep

// until a customer arrives.

sem\_t barberPillow;

// seatBelt is used to make the customer to wait until

// the barber is done cutting his/her hair.

sem\_t seatBelt;

// Flag to stop the barber thread when all customers

// have been serviced.

int allDone = 0;

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

  pthread\_t btid;

  pthread\_t tid[MAX\_CUSTOMERS];

  long RandSeed;

  int i, numCustomers, numChairs;

  int Number[MAX\_CUSTOMERS];

  // Check to make sure there are the right number of

  // command line arguments.

  if (argc != 4){

    printf("Use: SleepBarber <Num Customers> <Num Chairs> <rand seed>\n");

    exit(-1);

  }

  // Get the command line arguments and convert them

  // into integers.

  numCustomers = atoi(argv[1]);

  numChairs = atoi(argv[2]);

  RandSeed = atol(argv[3]);

  // Make sure the number of threads is less than the number of

  // customers we can support.

  if (numCustomers > MAX\_CUSTOMERS) {

    printf("The maximum number of Customers is %d.\n", MAX\_CUSTOMERS);

    exit(-1);

  }

  printf("\nSleepBarber.c\n\n");

  printf("A solution to the sleeping barber problem using semaphores.\n");

  // Initialize the random number generator with a new seed.

  srand48(RandSeed);

  // Initialize the numbers array.

  for (i=0; i<MAX\_CUSTOMERS; i++) Number[i] = i;

  // Initialize the semaphores with initial values...

  sem\_init(&waitingRoom, 0, numChairs);

  sem\_init(&barberChair, 0, 1);

  sem\_init(&barberPillow, 0, 0);

  sem\_init(&seatBelt, 0, 0);

  // Create the barber.

  pthread\_create(&btid, NULL, barber, NULL);

  // Create the customers.

  for (i=0; i<numCustomers; i++) pthread\_create(&tid[i], NULL, customer, (void \*)&Number[i]);

  // Join each of the threads to wait for them to finish.

  for (i=0; i<numCustomers; i++) pthread\_join(tid[i],NULL);

  // When all of the customers are finished, kill the

  // barber thread.

  allDone = 1;

  sem\_post(&barberPillow); // Wake the barber so he will exit.

  pthread\_join(btid,NULL);

}

void \*customer(void \*number) {

  int num = \*(int \*)number;

  // Leave for the shop and take some random amount of

  // time to arrive.

  printf("Customer %d leaving for barber shop.\n", num);

  randwait(5);

  printf("Customer %d arrived at barber shop.\n", num);

  // Wait for space to open up in the waiting room...

  sem\_wait(&waitingRoom);

  printf("Customer %d entering waiting room.\n", num);

  // Wait for the barber chair to become free.

  sem\_wait(&barberChair);

  // The chair is free so give up your spot in the

  // waiting room.

  sem\_post(&waitingRoom);

  // Wake up the barber...

  printf("Customer %d waking the barber.\n", num);

  sem\_post(&barberPillow);

  // Wait for the barber to finish cutting your hair.

  sem\_wait(&seatBelt);

  // Give up the chair.

  sem\_post(&barberChair);

  printf("Customer %d leaving barber shop.\n", num);

}

void \*barber(void \*junk) {

  // While there are still customers to be serviced...

  // Our barber is omnicient and can tell if there are

  // customers still on the way to his shop.

  while (!allDone) {

    // Sleep until someone arrives and wakes you..

    printf("The barber is sleeping\n");

    sem\_wait(&barberPillow);

    // Skip this stuff at the end...

    if (!allDone) {

      // Take a random amount of time to cut the

      // customer's hair.

      printf("The barber is cutting hair\n");

      randwait(3);

      printf("The barber has finished cutting hair.\n");

      // Release the customer when done cutting...

      sem\_post(&seatBelt);

    }

    else printf("The barber is going home for the day.\n");

  }

}

void randwait(int secs) {

  int len;

  // Generate a random number...

  len = (int) ((drand48() \* secs) + 1);

  sleep(len);

}