Ahmedabad
University

CSE332 : Operating System Lab

Section 2

Monsoon Semester 2021

Sakshi Shah - AU1940213

End Semester Examination - Code

date: 23th November 2021

Question 1 :

(A) Using Multi threading

- **Code :**

```
/*
--> AU1940213-Sakshi Shah
--> Operating System Lab - Section-2, Monsoon Semester 2021
--> End-Semester Examination
--> Date : 23rd November 2021
--> Question :  1

*/
#include <stdio.h> //standard input output library
#include <stdlib.h>
#include <sys/types.h> //for wait(),fork()
#include <unistd.h>    //for pipe(),fork(),read(),write(),provides access to posix()
#include <fcntl.h>   //for functions like pipe(), open()
```

```
#include <sys/wait.h> //for wait()
#include <errno.h>  //error handling
#include <string.h> //string function e.g strlen()
#include <sys/stat.h>
//struct library
#include <sys/stat.h>
#include<dirent.h>
#include <sys/dir.h>

#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t* cond = NULL;

int threads;
volatile int cnt = 0;

//  synchronize threads
void* foo(void* arg)
{
    // turn is a basically to identify a thread
    int turn = *(int*)arg;

    pthread_mutex_lock(&mutex);

    //condition to check
    if (turn != cnt) {

        pthread_cond_wait(&cond[turn], &mutex);
    }

    // it's a time to print turn can have
    // values starting from 0. Hence + 1
    printf("%d ", turn + 1);

    // determine which thread need to be scheduled now
    if (cnt < threads - 1) {
        cnt++;
    }
    else {
        cnt = 0;
    }
    pthread_cond_signal(&cond[cnt]);
    pthread_mutex_unlock(&mutex);
    }
    return NULL;

int main()
{

    pthread_t* tid;
    volatile int i;
    int* arr;

    threads=26;

    cond = (pthread_cond_t*)malloc(sizeof(pthread_cond_t)
                                   * threads);
    tid = (pthread_t*)malloc(sizeof(pthread_t) * threads);
    arr = (int*)malloc(sizeof(int) * threads);
    for (int i = 0; i < threads; i++) {
    if (pthread_cond_init(&cond[i], NULL) != 0) {
        perror("pthread_cond_init() error");
        exit(1);
    }
    }

    for (i = 0; i < threads; i++) {
    arr[i] = i;
    pthread_create(&tid[i], NULL, foo, (void*)&arr[i]);
    }


    for (i = 0; i < threads; i++) {
    pthread_join(tid[i], NULL);
    }

    return 0;
}

/*int main(void) {
  int x,y;
  int sum;

  pid_t pid = fork();

  if(pid > 0) { // in child process
    printf("Enter values \n");
    scanf("%d",&x);
    scanf("%d",&y);
    //pin = "4821\0"; // PIN to send
    //close(pipefds[0]); // close read fd
    //write(pipefds[1], x, 5); // write PIN to pipe
    //write(pipefds[1], y, 5); // write PIN to pipe
    wait(NULL);
    printf("calculating values..\n");
    sleep(2); // intentional delay
    printf("%d\n",24);
    exit(EXIT_SUCCESS);
  }

  if(pid = 0) { // in main process

    //printf ("A 1 a B 2 b C 3 c D 4 d E 5 e F 6 f G 7 g H 8 h I 9 i J 10 j K 11 k L 12 l M 13 m
    N 14 n O 15 o P 16 p Q 17 q R 18 r S 19 s T 20 t U 21 u V 22 v W 23 w X 24 x Y 25 y Z
    26 z"); ==array
return 0;
    //wait(NULL); // wait for child process to finish
    //close(pipefds[1]); // close write fd
    //read(pipefds[0], x, 5); // read PIN from pipe
    //read(pipefds[0], y, 5); // read PIN from pipe
    //sum = x +y;
    //close(pipefds[0]); // close read fd
    //printf("Parent received PIN '%d'\n", sum);
  }
  return EXIT_SUCCESS;
}
```

- **RR - Code :**

```
 /*
--> AU1940213-Sakshi Shah
--> Operating System Lab - Section-2, Monsoon Semester 2021
--> End-Semester Examination
--> Date : 23rd November 2021
--> Question :  2

*/
#include <stdio.h> //standard input output library
#include <stdlib.h>
#include <sys/types.h> //for wait(),fork()
```

```
#include <unistd.h>    //for pipe(),fork(),read(),write(),provides access to posix()
#include <fcntl.h>   //for functions like pipe(), open()
#include <sys/wait.h> //for wait()
#include <errno.h>  //error handling
#include <string.h> //string function e.g strlen()
#include <sys/stat.h>
//struct library
#include <sys/stat.h>
#include<dirent.h>
#include <sys/dir.h>

void main()  {

// variables for turn around time, average turn around time, wating time and average waiting
    time for each process
    int i,m,n,y;
    int sum=0,count=0,quant_time;
    int wait_time=0, tat=0, arrive_time[10], burst_time[10], temp[10];
    int num;
    float avg_wait, avg_tat;

    //input the number of processes
    printf(" Total Number of process to be executed in the system with Round Robin method :
    ");
    scanf("%d", &num);
    y = num;  //assign number to another variable


    for(i=0; i<num; i++)
    { //take infut for number of processes : arrival time and burst time
    printf("\nProcess [%d]\n", i+1);
    printf("Arrival Time of Process: ");
    //input arrival time
    scanf("%d", &arrive_time[i]);
    printf("Burst Time of the Process : ");
    scanf("%d", &burst_time[i]);
    //input burst time
    temp[i] = burst_time[i]; // store the burst time in temp array
    }

    printf("Enter the Time Quantum (q) for the RR scheduling for above processes: \t");
    scanf("%d", &quant_time);
    //input time quantum

    printf("\nProcess Id : \t Burst Time : \t TurnAroundTime \t Waiting Time ");
```

```
for(sum=0, i = 0; y!=0;)
{
if(temp[i] <= quant_time && temp[i] > 0) //when quanttime is no reaced contimue the
process
{
sum = sum + temp[i];
temp[i] = 0;
count=1;
}
else if(temp[i] > 0)
{
    temp[i] = temp[i] - quant_time;
    sum = sum + quant_time;
}
if(temp[i]==0 && count==1) //a process completed
{
    y--;
    printf("\nProcess Id[%d] \t     %d   \t\t %d\t\t\t %d",
    i+1,
    burst_time[i],
    sum-arrive_time[i],
    sum-arrive_time[i]-burst_time[i]);
    wait_time = wait_time+sum-arrive_time[i]-burst_time[i];
    tat = tat+sum-arrive_time[i];
    count =0;
}
if(i==num-1)
{
    i=0;
}
else if(arrive_time[i+1]<=sum)
{
    i++;
}
else
{
    i=0;
}
}

avg_wait = wait_time * 1.0/num;

avg_tat = tat * 1.0/num;

printf("\nAverage Turn Around Time : %f\n", avg_wait);
printf("Average Waiting Time : %f\n", avg_tat);
```

```
    //return 0;
    }
```

- **RR - modified Code :**

```
 /*
--> AU1940213-Sakshi Shah
--> Operating System Lab - Section-2, Monsoon Semester 2021
--> End-Semester Examination
--> Date : 23rd November 2021
--> Question :  2

*/
#include <stdio.h> //standard input output library
#include <stdlib.h>
#include <sys/types.h> //for wait(),fork()
#include <unistd.h>    //for pipe(),fork(),read(),write(),provides access to posix()
#include <fcntl.h>   //for functions like pipe(), open()
#include <sys/wait.h> //for wait()
#include <errno.h>  //error handling
#include <string.h> //string function e.g strlen()
#include <sys/stat.h>
//struct library
#include <sys/stat.h>
#include<dirent.h>
#include <sys/dir.h>
#include<stdio.h>
struct process
{
    int wait_time,AT,BT,TAT,PT, quantam;
};

struct process a[10];

int main()
{
    int n,temp[10],t,count=0,short_p;
    int quantam;
    float total_wait=0,total_TAT=0,avg_wait,avg_tat;
    printf("Enter total number of the processes to be executed under RR: ");
    scanf("%d",&n);
    printf("\nEnter the Arrival time , Burst time and Priority for each process : ");
```

```
for(int i=0;i<n;i++)
{
printf("\nProcess Id : [%d]\n", i + 1);
 printf("Arrival Time of Process: ");
scanf("%d", &a[i].AT);
printf("Burst Time of Process : ");
    scanf("%d", &a[i].BT);

    printf("Priority of the process: ");

    scanf("%d", &a[i].PT);

temp[i]=a[i].BT;
}
printf("Enter quantum time: ");
scanf("%d",&quantam);
a[9].PT=10000;

for(t=0;count!=n;t++)
{
short_p=9;
for(int i=0;i<n;i++)
{
    if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
    {
            short_p=i;
    }
}

a[short_p].BT=a[short_p].BT-1;
    if(a[short_p].BT==0)
{           count++;
    a[short_p].wait_time=t+1-a[short_p].AT-temp[short_p];
    a[short_p].TAT=t+1-a[short_p].AT;
    total_wait=total_wait+a[short_p].wait_time;
    total_TAT=total_TAT+a[short_p].TAT;

}
}

avg_wait=total_wait/n;
avg_tat=total_TAT/n;

// printing of the answer
printf("\nProcess ID  \tWait-Time \t Burst-Time \t Turnaround-Time \t\n");
for(int i=0;i<n;i++)
```

```
    {
    printf("\nProcees ID[%d]\t\t%d \t\t%d \t\t %d",i+1,a[i].wait_time,temp[i],a[i].TAT);
    }


    printf("\n\nAverage Turn Around Time : %f\n", avg_wait);
    printf("Average Waiting Time : %f\n", avg_tat);
    return 0;
}
```

Question 3

- **Code :**

```
/*
--> AU1940213-Sakshi Shah
--> Operating System Lab - Section-2, Monsoon Semester 2021
--> End-Semester Examination
--> Date : 23rd November 2021
--> Question :  3

*/
#include <stdio.h> //standard input output library
#include <stdlib.h>
#include <sys/types.h> //for wait(),fork()
#include <unistd.h>    //for pipe(),fork(),read(),write(),provides access to posix()
#include <fcntl.h>   //for functions like pipe(), open()
#include <sys/wait.h> //for wait()
#include <errno.h>  //error handling
#include <string.h> //string function e.g strlen()
#include <sys/stat.h>
//struct library
#include <sys/stat.h>
#include<dirent.h>
#include <sys/dir.h>
#include <pthread.h>
#include <semaphore.h>
#include "buffer.h" //file created


#define RAND_DIVISOR 100000000
#define TRUE 1
```

```
pthread_mutex_t mutex; //thread for mutex that helps achieve synchronization, in critical section - 1 process
sem_t full_slot, empty_slot; //2 semaphores signaling empty and full slots
buffer_item buffer[BUFFER_SIZE]; //buffer - buffer size managed in buffer.h file in header
int counter;
pthread_t thread_id;    //Thread ID
pthread_attr_t attr; //Set of thread attributes

void *producer(void *param); //declare producer thread
void *consumer(void *param); //declare consumer thread

void initializeData() {

  pthread_mutex_init(&mutex, NULL);
  sem_init(&full_slot, 0, 0);//full slot semaphore
  sem_init(&empty_slot, 0, BUFFER_SIZE); //empty slots semaphore
  pthread_attr_init(&attr);
  counter = 0; // in buffer
}

//Producer Thread
void *producer(void *param) {
  buffer_item item;

  while(TRUE) {

    int ran = rand() / RAND_DIVISOR;
    sleep(ran); //sleep
    item = rand();//item to store

    sem_wait(&empty_slot);//see if slot empty
    pthread_mutex_lock(&mutex);//mutex free
    if(insert_item(&item)) {
    fprintf(stderr, " Producer report error condition\n");
    }
    else {
    printf("Item produced by producer : %d\n", item);
    }
    pthread_mutex_unlock(&mutex);//mutex free
    sem_post(&full_slot); //semsignal
  }
}

//Consumer Thread
void *consumer(void *param) {
  buffer_item item;
  while(TRUE) {

    int ran = rand() / RAND_DIVISOR;
    sleep(ran); //sleep

    sem_wait(&full_slot);//sem lock
    pthread_mutex_lock(&mutex);//mutex lock
    if(remove_item(&item)) {//remove item
    fprintf(stderr, "Consumer report error condition\n");
    }
    else {
    printf("Item removed by consumer : %d\n", item);
    }
    pthread_mutex_unlock(&mutex);
    sem_post(&empty_slot);
  }
}
//producer work
int insert_item(buffer_item item) {
  if(counter < BUFFER_SIZE) {
//any free slot in buffer
    buffer[counter] = item;
    counter++;
    return 0;
  }
  else {
    return -1; //error that buffer is full
  }
}

//consumer work
int remove_item(buffer_item *item) {
  if(counter > 0) { //counter = number of items in queue
    *item = buffer[(counter-1)];
    counter--;
    return 0;
  }
  else {
    return -1; //return error that buffer empty
  }
}
int main(int argc, char *argv[]) {
  int i;
  if(argc != 4) {
    fprintf(stderr, "USAGE:./a.out (var1) (var2) (var3)\n");
  }
  int sleept = atoi(argv[1]); //sleep time
  int producernumber = atoi(argv[2]); // Number of producer threads
  int consumernumber = atoi(argv[3]); // Number of consumer threads
  initializeData();
  for(i = 0; i < producernumber; i++) {
    pthread_create(&thread_id,&attr,producer,NULL); //producer thread
    }
  for(i = 0; i < consumernumber; i++) {
    pthread_create(&thread_id,&attr,consumer,NULL); //consumer thread
  }

  /* METHOD 2 to implement
  printf("Enter the option to be executed: ");
    scanf("%d", &option);
    if (option ==1) {
        if ((mutex==1) && (empty_slots !=0)) {
                //producer();
                pthread_create(&thread_id,&attr,producer,NULL); //producer thread
        }
        else {
                printf("Buffer is full\n");
                }
   }
        if (option == 2){
   if ((mutex==1) && (full_slots !=0)) {
                //consumer();
                pthread_create(&thread_id,&attr,consumer,NULL); //consumer thread
        }
        else {
                printf("Buffer is empty\n");
                }
   }
        if (option == 3){
   exit(0);
   break;
        }
  */


  sleep(sleept); //sleep for sometime
  printf("Exit the program\n");
  exit(0);
}
/*
//Producer - Consumer Problem without semaphore
#include <stdio.h>
```

```
#include <stdlib.h>
int mutex=1;
int full_slots=0;
int empty_slots=10;
int x=0;
void producer() {
 mutex=mutex-1;
 full_slots = full_slots +1; //increase the number of slots
 empty_slots=empty_slots-1;//one empty slot decreases
 x=x+1;
printf("Item produced by producer : %d\n",x);
mutex=mutex+1;
}
void consumer() {
  mutex = mutex-1;
  full_slots=full_slots-1;
  empty_slots=empty_slots+1;
  printf("Item removed by consumer : %d\n",x);
  x=x-1;
  mutex=mutex+1;
}

int main() {
int option, i;
      printf("\n 1. Producer \n 2. Consumer \n 3. Exit \n");
      #pragma omp critical
      while (1) {
      printf("Enter the option to be executed: ");
      scanf("%d", &option);
      if (option ==1) {
            if ((mutex==1) && (empty_slots !=0)) {
                        producer();
            }
            else {
                        printf("Buffer is full\n");
                        }
    }
            if (option == 2){
    if ((mutex==1) && (full_slots !=0)) {
                        consumer();
            }
            else {
                        printf("Buffer is empty\n");
                        }
    }
            if (option == 3){
```

```
   exit(0);
   break;
         }
}
}
*/
```