Fast**National University of Computer & Emerging Sciences, Karachi  
Spring 2021 CS-Department  
Final Examination Solution  
08th June 2021, 12:30 pm – 02:00 pm**

|  |  |  |
| --- | --- | --- |
| **Course Code: CL 220** | **Course Name: Operating Systems Lab** | |
| **Instructor Name / Names: Tania Iram, Anaum Hamid, Safia, Rabia, Ali Fatmi** | | |
| **Student Roll No:** | | **Section No:** |
| **PaperBBBBBBBBBBBBBBBBBBBB** | | |

Instructions:

* Return the question paper.
* Read each question completely before attempting it. There are **5 questions and 4 pages**
* In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

**Time**: 90 minutes. **Max Marks:** 50 points

**Q1. Signal Handling Marks 5**

Complete the program that implements Alarm function. Program keep printing “Alarm is on” after each second five times, then it turns off for five seconds. When alarm is off it keeps printing “Alarm is off”. Implement an interrupt call which snoozes alarm and prints “Alarm is snoozed”. Implement a stop call that stops the alarm and prints “Alarm is stopped” & terminates the program.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #include<stdio.h>  #include<unistd.h>  #include<wait.h>  #include<signal.h>  #include<errno.h>  #include <stdlib.h>  int check=0;  void wakeup() {   |  | | --- | | for(int i=0;i<5;i++){  if(check==1){  check=0;  break;}  printf("Alarm is ringing\n");  sleep(1);} |   }  void snooze( int signum) {   |  | | --- | | printf("Alarm is snoozed\n");  check=1; |   } | void stop( int signum) {   |  | | --- | | printf("Alarm is stopped\n");  exit (signum); |     }  int main() {  signal(SIGALRM,wakeup);  signal(SIGINT, snooze);  signal(SIGTSTP, stop);  while(1){  alarm(5);  pause();  printf("Alaram is of\n");    }  return 0;    } |

**Q2. Shell scripting and Process Management and Communication Marks** 10

a)Write a program ‘priority.sh’ to open firefox with -4 priority

b) write a c program to send the name of the shell script in part a from parent process to child process using pipes. Child will then execute that shell script and exit.

|  |  |
| --- | --- |
| **C program**  #include<stdio.h>  #include<unistd.h>  #include<sys/types.h>  int main(int argc, char \*argv[])  {  int pipefd[2];  int pid;  char buffer[30];  pipe(pipefd);  pid = fork();  if(pid > 0) {  fflush(stdin);  printf("unamed\_pipe [INFO] Parent Process\n");  write(pipefd[1],"./priority.sh",30);  }  else if(pid == 0) {  sleep(5);  fflush(stdin);  printf("unamed\_pipe [INFO] Child Process\n");  read(pipefd[0], buffer, sizeof(buffer));  write(1,buffer, sizeof(buffer));  printf("\n");  execlp(buffer,buffer,NULL);  }  else {  printf("unamed\_pipe [ERROR] Error in creating child process\n");  }  if(pid > 0) wait();  return 0;  return 0;  } | **priority.sh**  sudo nice -n -4 firefox |

**Q3: Multithreaded Programming using Pthreads Marks 10**

Complete the code of a simple multithreaded program in C that create multiple threads by default attributeson Linux environment and each thread will print the table till 10 with its Id number. Performnecessary error checking where needed. In given Code Grey area. main has 1 and in thread has more than 2 lines missing.

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#define N 4

void \*child\_thread(void \*arg)

{

|  |
| --- |
| printf("This is thread #%ld\n", (long)arg);  int i;  for (i = 1; i <= 10; ++i) {  printf("%ld \* %d = %ld\n", (long)arg, i, (long)arg \* i);  }  pthread\_exit(NULL); |
|  |
|  |
|  |

}

int main()

{ pthread\_t my\_thread[N];

long id;

for(id = 1; id <= N; id++) {

int value = pthread\_create(&my\_thread[id], NULL, &child\_thread, (void\*)id); if(value != 0) {

printf("Error: pthread\_create() failed\n");

exit(EXIT\_FAILURE); }

} pthread\_exit(NULL);

}

**Q4: Multithreaded Programming using OpenMp Marks 15**

In stock market the shares of companies go up and down during a day. Write an OpenMp program for stock market shares update according to the given requirement:

* You need to initialize two arrays with random values.
* Array ‘a[8]’ will hold the opening price of stock for 8 companies.
* Array d[8] will be initialized with random values between one to ten. (use rand() function).
* Four openmp threads will update the values of two companies, once only, in each iteration after checking the contents of d[i] (criteria is increase the share value by d[i] value if value greater than 5 otherwise decrease by the value of d[i]).
* Display which thread update the value of which company
* Display the closing price of shares of each

|  |
| --- |
| #include <stdlib.h> //malloc and free  #include <stdio.h> //printf  #include <omp.h> //OpenMP  // Very small values for this simple illustrative example  #define ARRAY\_SIZE 8 //Size of arrays showing number of companies.  #define NUM\_THREADS 4 //Number of threads to use for vector addition or subtraction.  int main (int argc, char \*argv[])  {  // elements of arrays a and b will be added  // and placed in array c  int \* a;  int \* c;  int \* d;  int n = ARRAY\_SIZE; // number of array elements  int n\_per\_thread; // elements per thread  int total\_threads = NUM\_THREADS; // number of threads to use  int i; // loop index    // allocate spce for the arrays  a = (int \*) malloc(sizeof(int)\*n);  c = (int \*) malloc(sizeof(int)\*n);  d = (int \*) malloc(sizeof(int)\*n);  // initialize arrays a and b with consecutive integer values  // as a simple example  for(i=0; i<n; i++) {  a[i] = i+100;  }    // Additional work to set the number of threads.  // We hard-code to 4 for illustration purposes only.  omp\_set\_num\_threads(total\_threads);    // determine how many elements each process will work on  n\_per\_thread = n/total\_threads;    // Compute the vector addition  // Here is where the 4 threads are specifically 'forked' to  // execute in parallel. This is directed by the pragma and  // thread forking is compiled into the resulting exacutable.  // Here we use a 'static schedule' so each thread works on  // a 2-element chunk of the original 8-element arrays.  #pragma omp parallel for shared(a, c) private(i)  for(i=0; i<n; i++) {  d[i]=rand()%10;  if (d[i]>5)  c[i] = a[i]+d[i];  else  c[i]= a[i]-d[i];  // Which thread am I? Show who works on what company for this samll example  printf("Thread %d update shares of company%d\n", omp\_get\_thread\_num(), i);  }  printf("====Final values of shares in stock market====\n");  printf("i\ta[i]\t+or-\td[i]=\tc[i]\n");  for(i=0; i<n; i++) {  printf("%d\t%d\t\t%d\t%d\n", i, a[i],d[i], c[i]);  }    // clean up memory  free(a); free(c);    return 0;  } |

**Q3: Process Synchronization**  **Marks 10**

Complete the code for a synchronize X-ray treatment mechanism in the hospital using semaphore where there are 20 clients and 2 attendants, each needs to wait to get his appointment token/number. During waiting and treatment, patients feel sleepy so they sleep, weighting area takes 5 seconds sleep, and the attendant will usually take 20 sec for treatment, so patient need 20 seconds sleep. Main function is given Write the handler function code only.

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<string.h>

#include<unistd.h>

sem\_t appointment, waiting;

void handler(void\* ptr) {

|  |
| --- |
| int i=\*(int\*)ptr;  sem\_wait(&appointment);  printf(" patient %d waiting for token\n", i+1);  sleep(5);  sem\_post(&appointment);  printf(" patient %d get token\n", i+1);  sem\_wait(&waiting);  printf(" patient %d waiting for trun\n", i+1);  sleep(10);  sem\_post(&waiting);  printf("patient %d getting checkup finished\n", i+1); |

}

int main() {

int i,a[10];

pthread\_t patient[20];

sem\_init(&appointment,0,1);

sem\_init(&waiting,0,2);

for(i=0;i<20;i++) {

a[i]=i;

pthread\_create(&patient[i],0, (void\*)handler, (void\*)&a[i]);

}

for(i=0;i<20;i++) {

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

}

printf("Done");

//sem\_destory(&appointment);

//sem\_destory(&waiting);

return 0;

}