SOLVED BY PRINCE ALVI

Sample Paper
FINALTERM EXAMINATION
Fall 2022
CS604P – Operating Systems (Practical)

Time: 90 min
Marks: 40

Question No: 1 (Marks: 01) - Please choose the correct option

In Unix, which system call creates the new process that contains a copy of the parent address space?

A. Create
B. Fork
C. New
D. pipe

Question No: 2 (Marks: 01) - Please choose the correct option

The FCFS algorithm is highly problematic for _____
a) operating systems
b) multiprocessor systems
c) time sharing systems
d) multiprogramming systems

Question No: 3 (Marks: 01) - Please choose the correct option

If a parent process terminates_____, this results in the termination of all child processes; this is referred to as cascading termination.

A. Normally or abnormally
B. Abnormally
C. Normally
D. forcefully

Question No: 4 (Marks: 01) - Please choose the correct option

If process synchronization is not controlled in the dining philosopher problem, it will lead to

A. Deadlock
B. Starvation
C. Deadlock or Starvation
D. Deadlock and Starvation

SOLVED BY PRINCE ALVI

Question No: 5 (Marks: 01) - Please choose the correct option

Which of the following algorithms is used to avoid deadlock?

A. Dynamic Programming algorithm
B. Primality algorithms
C. Banker's algorithm
D. Deadlock algorithm

Question No: 6 (Marks: 01) - Please choose the correct option

Address generated by CPU is referred as
A. Physical address
B. Logical address
C. Page address
D. Segment address

Question No: 7 (Marks: 01) - Please choose the correct option

The instruction being executed, must be in _____
A. physical memory
B. logical memory
C. physical & logical memory
D. Registers

Question No: 8 (Marks: 01) - Please choose the correct option

When the CPU attempts to obtain a needed page exists in main memory (RAM), it is referred to as a
A. Page hit
B. Page miss
C. Page fault
D. Fault time

Question No: 9 (Marks: 01) - Please choose the correct option

Producer – consumer problem also known as
A. Reader-writer
B. Dinning-philosopher
C. Bounded-buffer
D. Writer – reader

Question No: 10 (Marks: 01) - Please choose the correct option

SOLVED BY PRINCE ALVI

Which of the following page replacement algorithms suffers from Belady's Anomaly?

A. Optimal replacement
B. LRU
C. FIFO
D. Both optimal replacement and FIFO

Question No: 11 (Marks: 01) - Please choose the correct option

LRU page - replacement algorithm associates with each page the _____

A. time it was brought into memory
B. the time of that page's last use
C. page after and before it
D. all of the mentioned

Question No: 12 (Marks: 01) - Please choose the correct option

Virtual memory is normally implemented by _____

A. demand paging
B. buses
C. virtualization
D. all of the mentioned

Question No: 13 (Marks: 01) - Please choose the correct option

In inter-process communication using pipe the _____ system call is used to create pipe between parent and child process.

A. segment()
B. page()
C. pipe()
D. fork()

Question No: 14 (Marks: 01) - Please choose the correct option

In Linux Terminal Window _____ command is used to compile the C Program.

A. vcc
B. gcc
C. vcc
D. dcc

SOLVED BY PRINCE ALVI

Question No: 15 (Marks: 01) - Please choose the correct option

---

In First Come First Serve Scheduling the turnaround time for each process can be calculated as_____.

A. Turnaround Time= Waiting Time * Burst Time
B. Turnaround Time= Waiting Time + Burst Time
C. Turnaround Time= Waiting Time / Burst Time
D. Turnaround Time= Waiting Time % Burst Time

Question No: 16 (Marks: 01) - Please choose the correct option

---

In Round Robin scheduling algorithm, we have equal _____for each process.

A. Time Stack
B. Time slice
C. Time controller
D. Time divider

Question No: 17 (Marks: 01) - Please choose the correct option

---

In Round Robin Scheduling, waiting time for each process can be calculated as _____.

A. Waiting Time = Turnaround Time+ Burst Time
B. Waiting Time = Turnaround Time- Burst Time
C. Waiting Time = Turnaround Time* Burst Time
D. Waiting Time = Turnaround Time/ Burst Time

Question No: 18 (Marks: 01) - Please choose the correct option

---

In Dining philosopher problem, continuous waiting for the fork by the philosopher will lead to _____situation.

A. paging
B. segmentation
C. deadlock
D. preempting

SOLVED BY PRINCE ALVI

Question No: 19 (Marks: 01) - Please choose the correct option

---

In _____when process enter the system, it must declare the maximum number of each resource type that it may need.

A. Deadlock Subtraction
B. Deadlock Avoidance
C. Deadlock Addition
D. Deadlock Controlling

Question No: 20 (Marks: 01) - Please choose the correct option

---

In Banker Algorithm data structure _____represents that process Pm can request maximum p instances of resource type Rn.

A. Max[n,m] = p
B. Max[m,n] = p
C. Max[p,n] = m
D. Max[n,p] = m

Question No: 21 (Marks: 01) - Please choose the correct option

---

In Linux Terminal Window_____command is used to print the path of current working directory

A. ts
B. cs
C. pwd
D. pss

Question No: 22 (Marks: 01) - Please choose the correct option

---

In Linux Terminal Window _____ command is used to change your current director to desktop.

A. ms Desktop
B. cc Desktop
C. cd Desktop
D. bc Desktop

Question No: 23 (Marks: 01) - Please choose the correct option

_____ command is used to create the new directory in the current working directory.

A. mkpir
B. mkdir
C. mkcir
D. mkbir

Question No: 24 (Marks: 01) - Please choose the correct option

---

The address generated by CPU is referred to as the _____.

A. Physical Address
B. Simple Address
C. Logical Address
D. Complex Address

Question No: 25 (Marks: 05)

Write a C program that provide the possible solution for producer – consumer problem using mutex and semaphore. In program make sure that producer doesn't try to put data in buffer when the buffer is full and consumer doesn't try to consumer data when the buffer is empty.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

#define BUFFER_SIZE 10 // Size of the buffer

int buffer[BUFFER_SIZE]; // Buffer to hold the items
int in = 0; // Index to add items to the buffer
int out = 0; // Index to remove items from the buffer

pthread_mutex_t mutex; // Mutex to protect access to the buffer
sem_t empty; // Semaphore to keep track of the empty slots in the buffer
sem_t full; // Semaphore to keep track of the filled slots in the buffer

void *producer(void *arg) {
   int item;

   while (1) {
      item = rand() % 100; // Generate a random item to produce
```

```
        sem_wait(&empty); // Wait for an empty slot in the buffer
        pthread_mutex_lock(&mutex); // Lock the buffer

        buffer[in] = item; // Add the item to the buffer
        printf("Producer produced item %d\n", item);
        in = (in + 1) % BUFFER_SIZE; // Increment the index

        pthread_mutex_unlock(&mutex); // Unlock the buffer
        sem_post(&full); // Signal that a slot in the buffer is filled
    }
}

void *consumer(void *arg) {
    int item;

    while (1) {
        sem_wait(&full); // Wait for a filled slot in the buffer
        pthread_mutex_lock(&mutex); // Lock the buffer

        item = buffer[out]; // Remove the item from the buffer
        printf("Consumer consumed item %d\n", item);
        out = (out + 1) % BUFFER_SIZE; // Increment the index

        pthread_mutex_unlock(&mutex); // Unlock the buffer
        sem_post(&empty); // Signal that a slot in the buffer is empty
    }
}

int main() {
    pthread_t producer_thread, consumer_thread;

    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);

    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);

    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);

    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);
```

```
    return 0;
}
```

Question No: 26 (Marks: 05)

Write a C program that implement the first readers-writers problem, requires that no reader be kept waiting unless a writer has already obtained permission to use the shared object. In other words, no reader should wait for other readers to finish simply because a writer is waiting.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

sem_t mutex, rw_mutex;
int read_count = 0;
int data = 0;

void *reader(void *arg) {
    int id = *(int *)arg;

    while (1) {
        sem_wait(&mutex); // Lock access to read_count
        read_count++;
        if (read_count == 1) {
            sem_wait(&rw_mutex); // Lock access to data if this is the first reader
        }
        sem_post(&mutex); // Unlock access to read_count

        printf("Reader %d read data: %d\n", id, data);

        sem_wait(&mutex); // Lock access to read_count
        read_count--;
        if (read_count == 0) {
            sem_post(&rw_mutex); // Unlock access to data if this is the last reader
        }
        sem_post(&mutex); // Unlock access to read_count

        sleep(rand() % 3); // Sleep for a random amount of time
    }
}
```

SOLVED BY PRINCE ALVI

```
void *writer(void *arg) {
    int id = *(int *)arg;

    while (1) {
        sem_wait(&rw_mutex); // Lock access to data
        data++;
        printf("Writer %d wrote data: %d\n", id, data);
        sem_post(&rw_mutex); // Unlock access to data

        sleep(rand() % 3); // Sleep for a random amount of time
    }
}

int main() {
    const int NUM_READERS = 5;
    const int NUM_WRITERS = 2;

    pthread_t reader_threads[NUM_READERS];
    pthread_t writer_threads[NUM_WRITERS];

    sem_init(&mutex, 0, 1);
    sem_init(&rw_mutex, 0, 1);

    int reader_ids[NUM_READERS];
    for (int i = 0; i < NUM_READERS; i++) {
        reader_ids[i] = i + 1;
        pthread_create(&reader_threads[i], NULL, reader, &reader_ids[i]);
    }

    int writer_ids[NUM_WRITERS];
    for (int i = 0; i < NUM_WRITERS; i++) {
        writer_ids[i] = i + 1;
        pthread_create(&writer_threads[i], NULL, writer, &writer_ids[i]);
    }

    for (int i = 0; i < NUM_READERS; i++) {
        pthread_join(reader_threads[i], NULL);
    }

    for (int i = 0; i < NUM_WRITERS; i++) {
        pthread_join(writer_threads[i], NULL);
    }

    sem_destroy(&mutex);
    sem_destroy(&rw_mutex);
```


For any query feel free to ask princealvi9222@gmail.com

```
    return 0;
}
```

Question No: 27 (Marks: 05)

Write a program in C to implement FCFS scheduling algorithm. Given the list of Processes and their burst time, calculate waiting time and turnaround time for each process. Also calculate average waiting time and average turnaround time. For example, the list of processes and their CPU burst time are as follows:

Processes	Burst Time
P0	11
P1	5
P2	7

Assume that all the processes arrive at time 0 in sequence P0, P1, P2

Solution:

```
#include <stdio.h>
int main(){
int n = 3 ; // number of processes
int bTime[3] ; // array for storing burst time for each process
int wTime[3] ; // array for storing wait time for each process
int tATime[3] ; // array for storing turnaround time for each process
bTime[0] = 11 ;
bTime[1] = 5 ;
bTime[2] = 7 ;
int avgWTime = 0 ;
int avgtATime = 0 ;
wTime[0] = 0 ;
// loop for calculating waiting time for each process
for(int i = 1; i<n; i++){
wTime[i] = wTime[i-1]+bTime[i-1] ; // calculating wait time for each process
avgWTime = avgWTime + wTime[i] ; }
// loop for calculating turnaround time for each process
for(int i = 0; i<n; i++){
tATime[i] = wTime[i]+bTime[i] ; // calculating turnaround time for each process
avgtATime = avgtATime + tATime[i] ;
}
avgWTime = avgWTime / n; // calculating average waiting time
avgtATime = avgtATime / n ; // calculating average turnaround time
// loop for displaying waiting time and turnaround times
for (int i =0; i<n; i++){
printf("P%i: " , i) ;
```

```
printf("Wait Time: %i\t", wTime[i]) ; // printing wait time for each process
printf("Turnaroung Time: %i\n", tATime[i]) ; // printing turnaround time for each
process
}
// displaying average waiting time and average turnaround time
printf("\nAverage Wait time: %i\n", avgWTime) ;
printf("\nAverage Wait time: %i\n\n", avgtATime) ;
} // end of main()#include <stdio.h>
int main(){
int n = 3 ; // number of processes
int bTime[3] ; // array for storing burst time for each process
int wTime[3] ; // array for storing wait time for each process
int tATime[3] ; // array for storing turnaround time for each process
bTime[0] = 11 ;
bTime[1] = 5 ;
bTime[2] = 7 ;
int avgWTime = 0 ;
int avgtATime = 0 ;
wTime[0] = 0 ;
// loop for calculating waiting time for each process
for(int i = 1; i<n; i++){
wTime[i] = wTime[i-1]+bTime[i-1] ; // calculating wait time for each process
avgWTime = avgWTime + wTime[i] ; }
// loop for calculating turnaround time for each process
for(int i = 0; i<n; i++){
tATime[i] = wTime[i]+bTime[i] ; // calculating turnaround time for each process
avgtATime = avgtATime + tATime[i] ;
}
avgWTime = avgWTime / n; // calculating average waiting time
avgtATime = avgtATime / n ; // calculating average turnaround time// loop for displaying
waiting time and turnaround times
for (int i =0; i<n; i++){
printf("P%i: " , i) ;
printf("Wait Time: %i\t", wTime[i]) ; // printing wait time for each process
printf("Turnaroung Time: %i\n", tATime[i]) ; // printing turnaround time for each
process
}
// displaying average waiting time and average turnaround time
printf("\nAverage Wait time: %i\n", avgWTime) ;
printf("\nAverage Wait time: %i\n\n", avgtATime) ;
} // end of main()
```

SOLVED BY PRINCE ALVI

Question No: 28 (Marks: 05)

Write the C program that take three parameters related to paging including page number, page offset and frame number and calculate the bits needed for logical and physical address space.

Solution:

```
#include <stdio.h>

int main() {
    int page_number, page_offset, frame_number;

    // Input the page number, page offset, and frame number
    printf("Enter the page number: ");
    scanf("%d", &page_number);

    printf("Enter the page offset: ");
    scanf("%d", &page_offset);

    printf("Enter the frame number: ");
    scanf("%d", &frame_number);

    // Calculate the number of bits needed for logical and physical address space
    int logical_address_bits = page_number + page_offset;
    int physical_address_bits = frame_number + page_offset;

    // Output the results
    printf("Number of bits needed for logical address space: %d\n", logical_address_bits);
    printf("Number of bits needed for physical address space: %d\n", physical_address_bits);

    return 0;
}
```

Question No: 29 (Marks: 05)

Write a c program to calculate Effective memory Access Time in paging. The performance measure of paging is the effective memory access time based upon Hit ratio, Time to Access Translation Look Aside Buffer T_{TLB}, and Memory Accss Time T_{mem} by using the below formula

Solution:

SOLVED BY PRINCE ALVI

```
#include <stdio.h>

int main() {
    double hit_ratio, t_tlb, t_mem, emat;

    // Input the hit ratio, TLB access time, and memory access time
    printf("Enter the hit ratio: ");
    scanf("%lf", &hit_ratio);

    printf("Enter the TLB access time: ");
    scanf("%lf", &t_tlb);

    printf("Enter the memory access time: ");
    scanf("%lf", &t_mem);

    // Calculate the effective memory access time
    emat = (hit_ratio * t_tlb) + ((1 - hit_ratio) * (t_tlb + t_mem));

    // Output the result
    printf("Effective memory access time: %lf\n", emat);

    return 0;
}
```

OR

```
#include<stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
double Tmem, Ttlb, hRatio, tEffective, mRatio;
printf("Calculating Effective Memory Access Time");
printf("\nEnter T(mem): ");
scanf("%lf", &Tmem);
printf("\nEnter T(TLB): ");
scanf("%lf", &Ttlb);
printf("\nEnter Hit Ratio: ");
scanf("%lf", &hRatio);
mRatio = 100 - hRatio;
tEffective = (hRatio/100)*(Ttlb+Tmem)+ (mRatio/100) * (Ttlb + 2*Tmem);
printf("\nTotal Effective Time for Memory Access: %lf", tEffective);
printf("\n\n\n");
system("pause");
}
```

SOLVED BY PRINCE ALVI

Question No: 30 (Marks: 05)

Write a C program using Inter-process communication between parent and child process through pipe. Child process should write your name through write() system call in the pipe. And Parent process read your name through read() system call and print your name on terminal widow through write() system call.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

#define PIPE_READ_END 0
#define PIPE_WRITE_END 1

int main() {
    int pipe_fd[2];
    pid_t pid;

    // Create the pipe
    if (pipe(pipe_fd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Fork the process
    pid = fork();

    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        // This is the child process
        close(pipe_fd[PIPE_READ_END]); // Close the unused read end of the pipe

        char name[] = "ChatGPT"; // My name to write to the pipe

        // Write my name to the pipe
        write(pipe_fd[PIPE_WRITE_END], name, sizeof(name));
        printf("Child process wrote '%s' to the pipe.\n", name);
```

```
        close(pipe_fd[PIPE_WRITE_END]); // Close the write end of the pipe
        exit(EXIT_SUCCESS);
    } else {
        // This is the parent process
        close(pipe_fd[PIPE_WRITE_END]); // Close the unused write end of the pipe

        char name[20]; // Buffer to read my name from the pipe

        // Read my name from the pipe
        read(pipe_fd[PIPE_READ_END], name, sizeof(name));
        printf("Parent process read '%s' from the pipe.\n", name);

        // Print my name to the terminal window
        write(STDOUT_FILENO, name, sizeof(name));

        close(pipe_fd[PIPE_READ_END]); // Close the read end of the pipe
        wait(NULL); // Wait for the child process to exit
        exit(EXIT_SUCCESS);
    }
}
```