Program Code TU856
Module Code CMPU 2017
CRN 24240 26463

# TECHNOLOGICAL UNIVERSITY DUBLIN

**Grangegorman**

---

# BSc in Computer Science.
# BSc in Computer Science (International).

**Year 2**

---

SEMESTER 1 EXAMINATIONS 2024/25

---

## Operating Systems 2

Internal Examiners

Mr. Denis Manley
Dr Paul Doyle

External Examiners

Dr. Colm O Riordan

Answer Questions 1 and any two others

Question 1 is worth 40 marks, all the rest are worth 30.

**1**

a) What are the four conditions that are required for Deadlock to occur? **(4 marks)**

b) Explain, using a suitable example, the steps required to push a node onto a stack. **(8 marks)**

c) Explain, using suitable examples, the steps required to add a node to an ordered linked list. **(10 marks)**

d)

i. Clearly explain, using suitable examples, if the following code will add two nodes to a queue data structure **(12 marks)**

ii. What would be the expected output using the printQueue function? **(6 marks)**

The function call and the parameters passed to the add function and the printQueue functions are:

```
QueueNode* headPtr = NULL; // initialize headPtr
QueueNode* tailPtr = NULL; // initialize tailPtr
char item; // char input by user

printf("%s", "Enter a character: ");
scanf("\n%c", &item);
add(&headPtr, &tailPtr, item);
printQueue(headPtr);
```

The code for the add function is:

```
// insert a node at queue tail
void add(QueueNode* *head, QueueNode* *tail, char value)
{
   QueueNode* newNode= malloc(sizeof(QueueNode));

   if (newNode != NULL) { // is space available
      newNode->data = value;
      newNode->nextPtr = NULL;

      // if empty, insert node at head
      if (*head == NULL) {
        *head = newNode;
      }
      else {
         (*tail)->nextPtr = newNode;
      }

      tail = newNode;

   }
   else {

      printf("%c not inserted. No memory available.\n", value);
   }
}
```

The code to print the Queue

```
// print the queue
void printQueue(QueueNode* currentPtr)
{
   // if queue is empty
   if (currentPtr == NULL) {
      puts("Queue is empty.\n");
   }
   else {
      puts("The queue is:");

      // while not end of queue
      while (currentPtr != NULL) {
     printf("%c|%p (%p) --> ", currentPtr->data,currentPtr->nextPtr,currentPtr);
         currentPtr = currentPtr->nextPtr;
      }

      puts("NULL\n");
   }
}
```

**2.**

a) Briefly describe the elements of a process control block (PCB) **(4 marks)**

b) In Linux a process is created using the *fork(), wait() and exec()* command. Explain, how the fork() command and wait() command are used to produce a child process in the following code: **(9 marks)**

denis.manley@apollo: ~/CT2/week7

```
int main(int argc, char **argv)
{
    printf("--beginning of program\n");
    int counter = 0;
    int second;

    pid_t pid = fork();

    if (pid == 0)
    {

        int max_c = atoi(argv[1]);

        int i,x;
        for (i = 0; i < 5; ++i)
        {
            printf("child process: counter=%d\n", ++counter);
            for(x=0; x<max_c;x++);
        }
    }
    else if (pid > 0)
    {
        int j,x, status;
        int  max_p = atoi(argv[2]);

        pid = wait(&status);
        for (j=0; j < 5; ++j)
        {
            printf("parent process: counter=%d\n", ++counter);
            for (x=0;x<max_p;x++);
        }
    }
    else
    {
        // fork failed
        printf("fork() failed!\n");
        return 1;
    }
    printf("--end of program--\n");
    return 0;
}
```

9,0-1 72%

c) In the code from part b what is the purpose of the command line arguments: 1000 and 10000 if the command line arguments are: *./fork 1000 10000*? **(5 marks)**

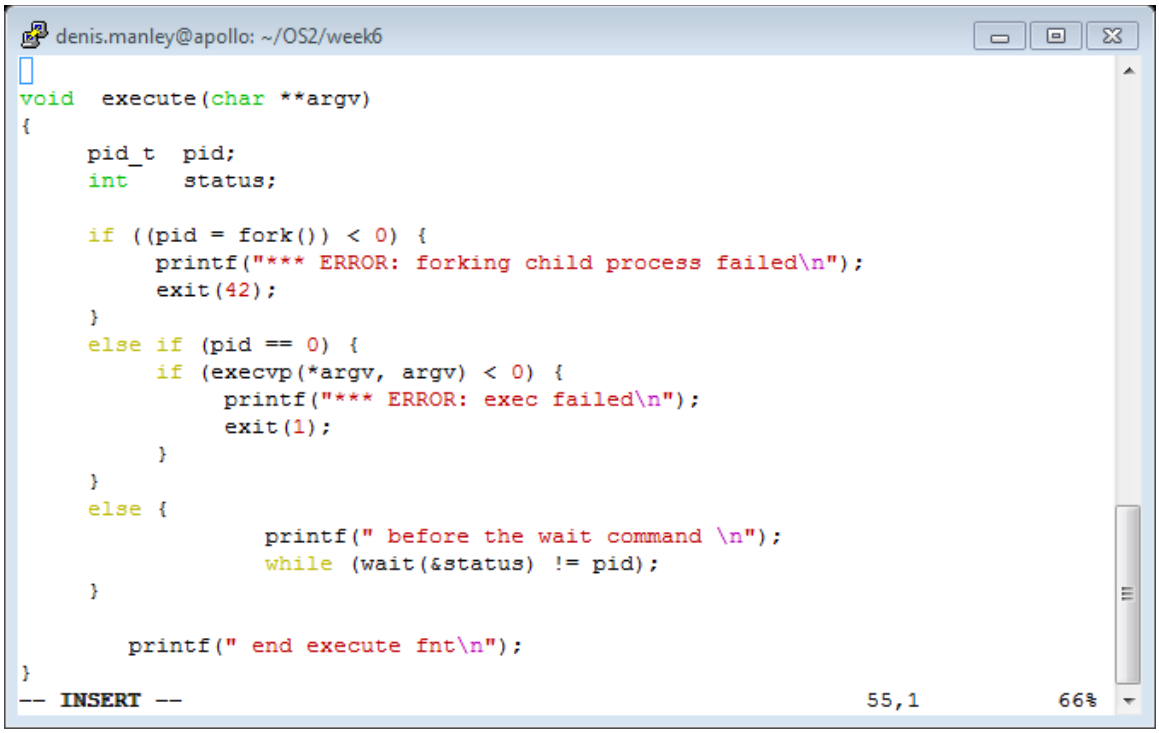
d) The execvp linux command has the following function prototype format

*int execvp(char *prog, char ** argv)*

Explain, using an example, what this function will do and what would data be stored in the prog parameter and the argv parameter if the commandline *cp file.c file2.c* .
(**4 marks)**

e)

The following code uses fork(), wait() and exec() functions. Explain what the code will do if the commandline arguments stored in arvg are *mv file1.c file2.c.* where mv is the linux rename function. **(8 marks)**



**3.**

a) Distinguish between single and multi-threading processes. **(4 marks)**

b) In C, a thread is created using the following code:

*int pthread_create(pthread_t *tidp, pthread_attr_t *attr, *start_rtn, void * arg)*

Clearly explain what each of the arguments in the thread create function mean.

**(8 marks)**

c) Explain, in your own words, the following code: **(10 marks)**

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

int value;
void *my_thread(void *param);             /* the thread */

main (int argc, char *argv[])
{
        pthread_t tid;          /* thread identifier */
        int retcode;


        if (argc != 4) {
                printf ("program exiting: incorrect number of command line arguments\n");
                exit(0);
        }
/*create Thread */
        retcode = pthread_create(&tid,NULL,my_thread,argv[3]);
        if (retcode != 0) {
                fprintf (stderr, "Unable to create thread\n");
                exit (1);
        }


        pthread_join(tid,NULL);

        printf("The value returned by the thread  is %d",value);
        printf ("\nThe end of the program\n");

        pthread_exit(0);
}


 /* explain that this thread does */
void *my_thread(void *param)
{
        int i = atoi(param);
        printf("I am the child thread passed the value %d \n",i);

        value = i*i*i*i;

        pthread_exit(0);
}
```

d) What would be the output of the code in part c if the following are input at the command prompt? **(6 marks)**

(a.) ./thread
(b.) ./thread 3 7 7
(c.) ./thread 3 7 5

e) What would be the two outcomes in the above program if the *pthread_join* command was removed and the command line input was *./thread 6 5 7*? Explain the reason for your answer. **(2 marks)**

**4.**

a) Explain, using an example, why it is critical to ensure that concurrency is carefully controlled for processes accessing the same data item; in other words the *race* problem. **(6 marks)**

b) How is the Test and Set approach used to prevent the race problem? **(2 marks)**

c) Explain, in detail, what the *signal* and *wait* threads are doing: **(12 Marks)**

if

TCOUNT = 5
COUNT_Limit = 6
count = 0

```
void *signal(void *t)
{
  int i;
  long my_id = (long)t;

  for (i=0; i < TCOUNT; i++) {
    pthread_mutex_lock(&count_mutex);
    count++;

    if (count == COUNT_LIMIT) {
      printf("Signal Thread: thread %ld, count = %d  Threshold reached. ",
            my_id, count);
      pthread_cond_signal(&count_threshold_cv);
    }

    pthread_mutex_unlock(&count_mutex);


    sleep(1);  // suspend thread
    }
  pthread_exit(NULL);
}
```

d) If *main()* has the following thread create calls:

```
int main(int argc, char *argv[])
{
  int i, rc;
  long t1=3, t2=1, t3=2;
  pthread_t threads[3];

  //create Three threads one to use the wait function and two for the signal function
  pthread_create(&threads[0], &attr, wait_fnt, (void *)t1);
  pthread_create(&threads[1], &attr, signal_fnt, (void *)t2);
  pthread_create(&threads[2], &attr, signal_fnt, (void *)t3);

  /* Wait for all 3 threads to complete */
  for (i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
  }
  printf ("Final value of count = %d.\n",count);
```

e) Give a sample output of this multithread program and explain your reasoning.

**(10 marks)**