DUBLIN INSTITUTE OF TECHNOLOGY
KEVIN STREET, DUBLIN 8.

DT228 BS. (Honours) Degree in Computer Science

DT282 BSc. (Honours) Degree in Computer Science (International)

Year 2

WINTER EXAMINATIONS 2016/2017

OPERATING SYSTEMS 2 [CMPU2017]

MR. DENIS MANLEY
DR. DEIRDRE LILLIS
MR. KEVIN FOLEY

WEDNESDAY 11TH JANUARY 1.00 P.M. – 3.00 P.M.

TWO HOURS

ANSWER QUESTIONS 1 AND ANY TWO OTHERS

QUESTION 1 IS WORTH 40 MARKS, ALL THE REST ARE WORTH 30.

1

a) Describe how the Little Man Computer (LMC) performs the Fetch and execute cycle **(8 marks)**

b) Illustrate the difference between *direct*, *single* direct and *double* direct pointers of an *i-node* control block. **(6 marks)**

c) Explain, using a suitable example, how you would pass a variable pointer to a function by reference. **(4 marks)**

d) Explain, using diagrams or otherwise , how you would:
add a node to the middle of an ordered link list

or

delete a node from the middle of an ordered link list. **(12 marks)**

e) Explain, in detail, if the following code will successfully push a node onto a stack.

//You can assume that the address of a pointer to the node at the top of the stack is passed to the function: push (&stackpointer, 5)

```
*
*   // insert a node with data value 5 onto the top of a stack
*
*   void function1(StackNodePtr *topPtr, int info)
*   {
*     StackNodePtr newPtr = malloc(sizeof(StackNode));
*
*
*     if (newPtr != NULL) {
*       newPtr->data = info;
*      newPtr->nextPtr = *topPtr;
*       topPtr = newPtr;
*     }
*     else { // no space available
*       printf("%d not inserted. No memory available.\n", info);
*     }
*   }
    *       }
        }
```

(10 marks)

2

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

b) In Linux a process is created using the *fork()* command. Explain the steps that occur if a *fork()* command is called **(6 marks)**

c) Explain, using a suitable example, exactly how the *wait()* functions works **(8 marks)**

d) the execvp linux command has the following format

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

Explain what is stored in the *two* parameters (*prog* and *argv*) if the following command line arguments are passed to the function: *cp file1.c file2.c* **(4 marks)**

e) The fork(), wait() and exec() functions can be used to run a new process:

Explain, in detail, what the following code does at the places indicated **(8 marks)**

```
if ((pid = fork()) < 0) {                    /*what happens here    */
    printf("*** ERROR: forking child process failed\n");
    exit(42);
}
else if (pid == 0) {


    if (execvp(*argv, argv) < 0)             /* what happens here.*/
        printf("*** ERROR: exec failed\n");
        exit(1);
    }
}
else {
    while (wait(&status) != pid)             /* what happens here.  */
        ;
}
```

3

a) Explain, using an example how you would map the logical address of a process to its physical address **(5 marks)**

b) What do you understand by the word trashing as it applies to virtual memory **(3 Marks)**

c) Describe what is the purpose of each field in the page map table of a virtual; memory system: status field, modified field, reference field and page frame field. **(8 marks)**

d) Page swapping is an essential element of virtual memory: two page swapping algorithms are the *First In First Out (FIFO)* and *Least Recently Used (LRU)* algorithms. Using, a suitable example, distinguish between each algorithm. **(10 marks)**

e) A *cache* improves the speed of processor access to instructions and data. Explain how the cache improves the speed of virtual memory page swapping. **(4 marks)**

4:

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

b) Two ways to prevent the race problem are Test and Set and Wait and Signal. Distinguish between each approach **(4 marks)**

c) Explain, in detail, what the following two threads are doing. **(12 Marks)**

```
// what does this thread do
void * signal(void *t)
{
for (i=0; i < 3; i++) {
 /* what is happening here */
   pthread_mutex_lock(&count_mutex);
   count++;
if (count == 4) {
        /* what is happening here */
    pthread_cond_signal(&count_threshold_cv);

    }

printf("inc_count(): thread %ld, count = %d, unlocking mutex\n",
            my_id, count);
/* what is happening here */
pthread_mutex_unlock(&count_mutex);
sleep(1);
pthread_exit();
}
```

```
// what does this thread do

void *wait(void *t){

/*  what is happening here */
   pthread_mutex_lock(&count_mutex);
    while (count < COUNT_LIMIT) {
      /* what is happening here */
       pthread_cond_wait(&count_threshold_cv, &count_mutex);
        count += 125;
       printf("watch_count(): thread %ld count now = %d.\n", my_id, count);
    }
   /* what is happening here */
  pthread_mutex_unlock(&count_mutex);
   pthread_exit();
}
```

d) A third method to prevent the race problem is by the use of *semaphores*. The following is an algorithm using semaphores to solve the bounded buffer producer consumer problem. Explain, using suitable examples, if can will solve the problem or if it may result in system deadlock **(8 marks)**

Assume the initial conditions are:

Empty: = 5; full: = 0; mutex: = 1

Producers Algorithm:

{P x > 0; x = x - 1);
(V x = x + 1}

Produce data	P(mutex)
P(mutex)	P(full)
P(empty)	# BEGIN CRITICAL REGION
#begin critical region	*read item from buffer*
write item into buffer	V(mutex)
V(mutex)	#END CRITICAL REGION
#END CRITICAL REGION	V(empty)
V(full)	Consume data