Computing Science 300

Quiz 1 SOLUTION
Summer 2018

1. Consider a computer system that includes a single 800 GB internal hard disk, 1 GB of RAM and 1 DMA. The DMA is used to transfer data and instructions between RAM and internal disk. It can transfer up to 100Mb at a rate of 10Mb/ms. The DMA setup takes 0.5 ms. Time taken by context switches and ISRs can be ignored. State any assumptions you make**.**

a) **[5 points]** Explain briefly what happens during the DMA setup.

- **During DMA setup the registers in the DMA hardware are initialized.**
- **The CPU sends values to the DMA that will be placed in the control registers of the DMA to initialize them. These values tell the DMA**
- **Which memory should be copied, how much memory should be copied,**
- **Where the block of memory to be copied begins**
- **Where the block of memory that the DMA should copy into begins.**

b) [5 points] Explain briefly what happens during the DMA setup.

The size of the block (500Mb) is larger than the size of the block the DMA can transfer it will be necessary to set up the DMA more than once

$$\frac{(size\ of\ block\ of\ data\ to\ be\ transferred)}{(\max size\ of\ block\ DMA\ can\ transfer} = \frac{500Mb}{100Mb} = 5$$

.

5 transfers are needed
Each transfer takes 10 ms (10ms * 10 Mb/ms = 100 Mb)
The DMA must be set up for each of the 5 transfers 0.5 ms for each transfer
5 transfers * (10ms / transfer + 0.5 ms / transfer) = 5 * 10.5 ms = 52.5 ms

c) [10 points] Give a step by step description of what happens when a 10Mb block of data is transferred from the hard disk to the RAM using the DMA. Assume that the CPU busy waits while IO is being performed.

A 10Mb block of data can be transferred in one DMA transfer

- **The process sets up the DMA (sets the values in the registers).**
- **The DMA transfers the data**
- **While the DMA transfers the CPU executes a busy waiting loop**
- **Each time through the loop the CPU will ask the DMA if it has finished (polls a register in the DMA to see if has been set to done yet). Last time through the loop the CPU sees the transfer is complete.**
- **If the DMA is done, the CPU will continue on the next instruction in the process**

2. The C code snippet shown below is a small chunk of the code extracted from a larger application. This code snippet shows additional processes being created. The process that includes the C code snippet creates a child and a grandchild. Answer each of the questions below. **Explain each of your answers**

```
1     parentPID = getpid();
2     pid = fork();
3     if ( pid == 0 )
4     {
5           printf("I am dog %d\n ", getpid());
6           if( (pid = fork() )  == 0 )
7                 {
8                       printf("I am dragon %d \n", getpid());
9                       sleep(15);  /* wait for 15 seconds */
10                      exit(1);
11                }
12                else if( pid > 0 )
13                {
14                      printf("who am I?\n" );
15                      waitpid( pid, &status, 0); /* blocking call
*/
16                      exit(1);
17                }
18                else
19                {
20                      printf("error\n");
21                }
22    }
23    else
24    {
25                printf("Am I dog or am I dragon?\n");
26    }
27    }
```

a) **[5 points]** Are both the child and the grandchild created within the code snippet? Which line of code, if any creates the child process? Which line of code, if any creates the grandchild process?

- **Both the child and the grandchild are created within the code snippet.**
- **The child is created in line 2,**
- **The grandchild is created in line 6.**
- **Line 2 begins by executing fork(). This creates a child process consisting of**
  - **a copy of the parent process image in memory (RAM)**
  - **a copy of the parent process control block**
- **Line 6 is part of the code executed by the child process. So executing the fork() in line 6 creates a new process that is created by the child (grandchild of the parent).**
  - **a copy of the child process image in memory (RAM)**
  - **a copy of the child process control block**
- **The newly created copy process is the child of the child process and the grandchild of the original parent process.**

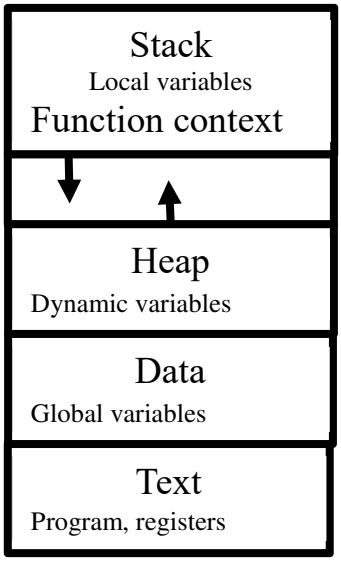
b) **[6 points]** Which lines in the provided code are executed only by the grandchild process? Which lines of the provided code are executed only by the parent process? Is the process that prints "I am a dog" the parent process, the child process or the grandchild process?

The groups of line numbers given include the lines containing brackets, omitting the brackets from the lines included will not result in points being deducted.

- **The lines executed ONLY by the grandchild are 7-11 (7, 11 are {})**
- **The lines executed ONLY by the parent process are lines 1 and 23-26 (23 and 26 are {})**
- **The line that prints "I am a dog" is line 5, it is printed by the child.**
- **Lines 2 and 6 include a call to fork(). The return value of the fork() is tested by both the child and parent on line 3 and by both grandchild and child on line 6. Since lines, 2,3, and 6 ,are executed by more than one process they are not part of the ranges given above.**
- **If the value returned by fork() in line 6 is 0 (the process is a grandchild) execution will continue to the call to exit() when the process is terminated. So the last line executed is 11**
- **The block of code entered in line 4 is complete in line 22 is for the child/grandchild. The parent code continues after the child code ends. (lines 23-26)**

c) **[6 points]** The code snippet above calls fork( ) to create a new process. When does fork( ) return zero? When fork returns a positive integer? What does that integer represent?
When fork() returns 0 to a process it indicates that that process is the child
When fork() returns a positive integer it indicates that the process is the parent
If the integer is returned to the parent and is > 0 it indicates the process id of the child
If the integer is negative it represents error condition information
If the integer is 0 it indicates the process receiving a value of 0 is the child

d) **[10 points]** Consider a process created by fork(). When the process is created the memory image and the process control block of the process are both copied. Draw a diagram of the image of a C program in memory. Label each portion of the image and briefly explain what is contained in each portion of the image. For example the code is contained in the text portion of the image.



e) **[6 points]** What is the purpose of the functions in lines 15, 16 and 1? Explain briefly what the function does and why it is needed.
Line 1: getpid(): get the process ID (pid) of the process calling the function
Line 15: waitpid(): Wait until a child process, or a child process with a specified pid to finish
Line 16: exit(): Terminate the process cleanly

f) **[6 points]** A process sets a local variable to have a value 3. The process then creates a new process using fork( ), Does the local variable in the child process have the value 3 immediately after returning from the call to fork()? Briefly explain why. Will the local variable always have the same value in the parent process and the child process? Briefly explain why.

Yes initially the value of the local variable in the child is 3. Creating a child process creates an exact copy of the memory image including the stack portion holding the local variables. Thus when we look at the value of single variable inside the memory image it will have the same value. However, any changes the parent makes to the value of the local variable after the child has been created will affect only the parent process (changes will only be made in the parent's memory image, not in the child's memory image). Similarly any changes the child makes to the value of the local variable will affect only the child. Therefore the value of the global variable will not always be the same in the parent and the child.

3. Consider a system that uses a 32-bit word. This system uses a 64 Mbyte cache memory on the CPU that has a 128 byte cache slot size. The system has of the 4096 Mbyte of RAM memory. For this problem assume 1 byte = 8 bits and 1Mbyte=2^{20} bytes and 1Kbyte = 2^{10} bytes. When a cache line is placed in the cache, the following rules are used.

    i. There are M cache lines in this system. The first cache line in the cache is cache line 0, the next is cache line 1, and so on. The last cache line is cache line M-1. All cache lines are the same size.)
    ii. There are N cache slots in this system. The first cache slot is cache slot 0, the next is cache slot 1, and so on. The last cache slot is cache slot N-1. All cache slots are the same size.
    iii. Consider that the cache is divided into P groups of 8 consecutive cache slots. (N=P*8)
    iv. Cache line K can be placed into any of the 8 cache slots in group Z=K%P
    v. If all cache slots in group Z are empty cache line K in group Z is placed in the first slot of group Z
    vi. If some cache slots in group Z are empty cache line K in group Z is placed in the empty slot of group Z with the lowest slot number.
    vii. If all cache slots in group Z are full the cache line will be placed in slot Q. Slot Q (in group Z) was last accessed before all other slots in group Z were last accessed.

a) **[8 points]** What is a cache line? How many cache lines would there be in this computer system? Where would the cache lines be located? How many cache slots would there be in this computer system?

- **The cache line is located in the RAM memory.**
- **A cache line is one of many equal sized sections of the RAM memory.**
- **The cache line is the same size as the cache slot, so each cache line is 128 bytes**
- **The size of the entire RAM memory is 4096 Mbyte = $2^{12}*10^{20}= 2^{32}$ bytes. If we divide the size of the RAM memory by the size of the cache line we will find the number of cache lines $2^{32}/128 = 2^{25}$ M=2^{25}**
- **The size of the entire cache memory is 64 Mbyte = $2^{6}*10^{20}= 2^{2}$ bytes. If we divide the size of the cache memory by the size of the cache line we will find the number of cache lines $2^{26}/128 = 2^{19}$ M=2^{19}**

b) **[6 points]** What is the purpose of a mapping function? For the system described above what would the mapping function be?

The purpose of the mapping function is to define the rules that determine which cache slots each cache line may be placed in.

The mapping function for this system is:

- **Cache line K can be placed into any of the 8 cache slots in group Z=K%P**
- **If all cache slots in group Z are empty cache line K in group Z is placed in the first slot of group Z**
- **If some cache slots in group Z are empty cache line K in group Z is placed in the slot of group Z with the lowest slot number.**

OR

- **list items from statement of problem items Iii –vi (no deduction for omitting iii)**

c) [6 points] What is the purpose of a replacement policy? For the system described above what would the replacement policy be?

The purpose of the replacement policy is to define the rules that determine which of the MAPPED cache slots to place the cache line into.

The replacement function for this system is:

- **If all cache slots in group Z are full the cache line will be placed in slot Q. Slot Q (in group Z) was last accessed before all other slots in group Z were last accessed.**

d) [8 points] Which cache slots could hold cache line 1234.

Number of slots N=$64*2^{20}/128=2^{19}$ (from part a)

P = N/8 = 2^{16} is the number of groups of 8 lines.

1234% 2^{16}= 1234 group 1234,

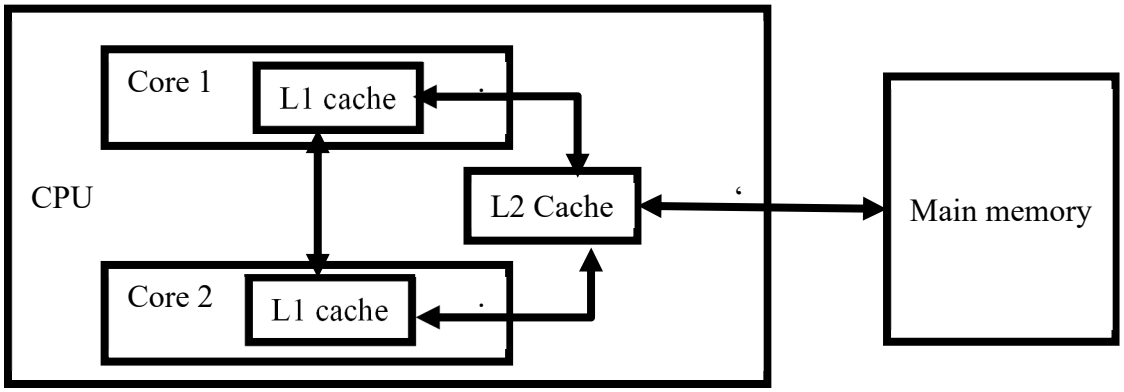
1234*8=9872 so slots 9872, 9873, 9874, 9875, 9876, 9877, 9879 and 9880

e) [5 points] Assume that each word in the RAM memory has an address. How many addresses are needed to refer to all words in the RAM memory?

4096 * 2^{20} bytes RAM / 4 bytes per word = 2^{30} words
1 address for each word means 2^{30} addresses

f) **[7 points]** Draw a diagram that shows how L1 and L2 cache might be configured in a multicore CPU with four cores (based on examples in our class notes or text)? What does the acronym NUMA stand for? How does it relate to cache in multicore CPUs?

Any reasonable diagram will receive points. I have shown an example below



The diagram should show

- **at least two levels of cache**
- **at least one of the levels of cache should be shared between cores**
- **diagram must show cache in CPU, main memory outside CPU**

NUMA stands for non-uniform memory access

A simple example is the easiest way to explain how NUMA relates to cache in multicore CPU's One example is shown in the diagram above. Any memory access from one of the cores may have a different (non the same = non-uniform) memory access time depending

- **Which cache data must be accessed from**
- **If the data needs to be accessed from multiple cores**