Eddie Gurnee
COSC 592
Operating Systems

Chapter 3 Reading Questions:

3.8) Describe the differences among short-term, medium-term, and long-term scheduling.

Response:

- short-term (CPU scheduler): selects a process from the job pool in memory that is ready to execute and allocates the CPU to one of them
- medium-term (does not have an affectionate name): in between the short and long term (sometime known as the 'just right' or 'goldilocks' term), this mechanism implements a swap space to hold the process state from memory when the process is not needed, then swaps it back into memory when memory is available. This is also the thing that my Mac has no space for anymore, so my computer crashes every other moment.
- long-term (job scheduler): selects processes from the pool of submitted and waiting processes and loads them into memory for execution

The primary distinction between the schedulers lie in the frequency of execution. The short-term selects a new process for the CPU frequently, while the long-term scheduler operates much less frequently, where minutes may separate the creation of one new process and the next.

3.9) Describe the actions taken by a kernel to context-switch between processes.

Response: To context-switch between processes, the current context of the process running on the CPU must be saved (the context being the value of the CPU registers, the process state, and memory-management information), which is known as a state save, and then a state restore (which is simply loading all of the data stored in a state save to the CPU) of the context the user is hoping to switch to.

3.14) Using the program in Figure 3.34, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

   /* fork a child process */
   pid = fork();

   if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
   }
   else if (pid == 0) { /* child process */
      pid1 = getpid();
      printf("child: pid = %d",pid); /* A */
      printf("child: pid1 = %d",pid1); /* B */
   }
   else { /* parent process */
      pid1 = getpid();
      printf("parent: pid = %d",pid); /* C */
      printf("parent: pid1 = %d",pid1); /* D */
      wait(NULL);
   }

   return 0;
}
```

Figure 3.34 What are the pid values?

Response:

- A = 0
- B = 2603
- C = 2603
- D = 2600

As `pid` is the value that is generated by `fork()`, the child should have 0 and the parent should have the child's process id, while `getpid()` will return the current process id to `pid1` for the child and parent, respectively.

3.17) Using the program shown in Figure 3.35, explain what the output will be at lines X and Y.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define SIZE 5

int nums[SIZE] = {0,1,2,3,4};

int main()
{
int i;
pid t pid;

   pid = fork();

   if (pid == 0) {
      for (i = 0; i < SIZE; i++) {
         nums[i] *= -i;
         printf("CHILD: %d ",nums[i]); /* LINE X */
      }
   }
   else if (pid > 0) {
      wait(NULL);
      for (i = 0; i < SIZE; i++)
         printf("PARENT: %d ",nums[i]); /* LINE Y */
   }

   return 0;
}
```

Figure 3.35 What output will be at Line X and Line Y?

Response:

- Line X = outputs the word 'CHILD' followed by the numbers inside of 'nums' (0..4) multiplied by the negation of numbers 0 to 5 (exclusive). Effectively it will look like it is counting by the negation of the perfect squares (0, -1, -4, -9, -16)
- Line Y = outputs the word 'PARENT' followed by the numbers inside of 'nums' (0..4).

I feel that it should be remarked that at the point of `fork()` executing there are effectively two arrays called 'nums', one in each process. These arrays are not synchronized, so changes made in one (specifically the changes made in the 'nums' of the child process) are not reflected in both.