Robert Weischedel
CS 6460
2/1/18

Homework 1

2.13 Describe at least two general methods for passing parameters to the operating system. (Hint: think about draw_me() in Assignment 1; how did the various syscalls that you used move data between your application and the kernel?)

1. You can pass parameters in registers.
2. You can make the registers pass starting addresses of blocks of parameters.
3. The parameters can also be placed, or pushed, onto the stack by the program, and popped off the stack by the operating system.

2.14 Describe how you could obtain a statistical profile of the amount of time spent by a program executing different sections of its code. Discuss the importance of obtaining such a profile.

You could cause periodic timer interrupts and monitor what sections of code are currently executing when the interrupts occur. The statistical profile of which sections of code are active should be consistent with the time spent by the program in different sections of its code. Once the statistical profile has been obtained, you could then optimize those sections of code that are consuming more of the CPU resources.

3.9 Describe the actions taken by a kernel to context-switch between processes. (A few sentences; don't go nuts.)

Context switching between kernel threads first requires you to save the value of the CPU registers from the thread being switched and then restoring the CPU registers of the new thread being scheduled.

3.12 Including the initial parent process, how many processes are created by the program shown below:

```
#include <stdio.h>
#include <unistd.h>

int main() {
```

```
fork();
fork();
fork();
return;
}
```

For this program you would have a total of 8 processes created.

3.17 Using the program below, 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;
}
```

Since the child is just a copy of the parent, any changes the child will make will only occur in its particular copy of the data and the parent won't be affected. Because of this, the values outputted by the child while at line X will be: 0, -1, -4, -9, -16. While the values output by the parent at line Y will be: 0, 1, 2, 3, 4.