Tutorial 3

1 Operating Systems Fundamentals — Paper and Pencil

1. What are the two main functions of an operating system?
2. What is multiprogramming? What is the primary purpose of multiprogramming?
3. Most operating systems are layered, what is the advantage of having a layered approach to operating system design?

2 Input and Output — Paper and Pencil

2.1 DMA

In virtually all systems that include a DMA (direct memory access) module, this runs at higher priority than the processor when accessing memory.

1. What is meant by 'higher priority' in this context?
2. Why is that a sensible policy for the DMA module?
3. Assume that a DMA is transmitting characters to main memory from an external device which is transmitting at a rate of 9600 bps (bits per second). The processor can fetch instructions at a rate of one million instructions per second. Estimate by how much the processor is slowed down by the DMA.

3 Programs and Processes in Linux

Consider the C program at the end of this section.

1. Compile and run this program. (You should know how to compile this program as a single command line.)
2. Run the program to start two or three calculators. Then open a new terminal and use top or pstree -p to find out:
   - The PID of each calculator process (make sure this matches what the program itself prints).
   - The PID of the process running the process_create program.
   - The PPID (parent process identifier) of each of the calculator processes.
3. Comment out the wait() line from the process_create program. Re-compile and re-run. What do you observe? Again, use top or pstree -p to determine the parent process identifier of each of the calculator processes.
4. Modify the process_create program to start instances of the “heart beat” program from tutorial 1 instead of calculators.
Make sure you are reasonably sure what each line of this program does. Use the command `man 2` to find out about `fork()`, `execvp()` and `wait()` if necessary.

```c
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int numprocs, i, status;
    printf("How many processes would you like to create?\n");
    scanf("%d", &numprocs);
    printf("Number of child processes that will be created: %d.\n", numprocs);
    for (i = 0; i < numprocs; ++i) {
        int pid = fork();
        if (pid < 0) {
            fprintf(stderr, "Could not create a new process.\n");
            exit(-1);
        } else if (pid == 0) {
            execvp("xcalc", "xcalc", NULL);
            /* This is the child process */
        } else {
            /* This is the parent process */
            printf("PID of the child process that has been created: %d.\n", pid);
        }
    }
    for (i = 0; i < numprocs; ++i) {
        int pid = wait(&status);
        printf("PID of process that just exited: %d.\n", pid);
    }
    return 0;
}
```

Olav Beckmann, Imperial College London, October 31, 2005