

Lectures

- In the lecture on February, 20th we will start with the chapter on System Structures (Chapter 2). In the lecture on February, 23rd, we will start with Chapter 3 from the course book: Process Concepts.
- The 2nd mandatory assignment is already available. Expect a deadline in the middle of March.
- Note again, that you find more exercises including solutions here :
<http://codex.cs.yale.edu/avi/os-book/OS9/practice-exer-dir/index.html>
- There are no tutorials in week 08.

Tutorial Session (Week 09)

- Prepare for the Tutorial Session in Week all exercises not discussed so far, and in addition:

Chapter 2:

2.2, 2.4, 2.5, 2.6, 2.7, 2.9, 2.10, 2.11

Chapter 3:

- 3.1 Describe the differences among short-term, medium-term, and long-term scheduling.
- 3.2 Describe the actions taken by a kernel to context-switch between processes.
- 3.4 Explain the role of the init process on UNIX and Linux systems in regards to process termination.
- 3.5 Including the initial parent process, how many processes are created by the program shown in the following program:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    /* fork a child process */
    fork();
    /* fork another child process */
    fork();
    /* and fork another */
    fork();
}
return 0;
```

3.6 Explain the circumstances when the line of code marked `printf("LINE J")` in the Figure below is reached.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid;
    /* fork a child process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
        printf("LINE J");
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
    }

    return 0;
}
```

3.7 Using the program in Figure below, 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 */
        A: execlp("/bin/ls", "ls", NULL);
        B: printf("LINE J");
    }
    else { /* parent process */
        C: /* parent will wait for the child to complete */
        D: wait(NULL);
        printf("Child Complete");
    }

    return 0;
}
```

```
}
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;
}
```

- 3.8 Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.
- 3.9 Consider the RPC mechanism. Describe the undesirable consequences that could arise from not enforcing either the "at most once" or "exactly once" semantic. Describe possible uses for a mechanism that has neither of these guarantees.
- 3.10 Using the program shown in Figure 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;
}
```

3.11 What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

- * Synchronous and asynchronous communication
- * Automatic and explicit buffering
- * Send by copy and send by reference
- * Fixed-sized and variable-sized messages