

Homework 5

*Reading: Tannenbaum, Sections 2.3, 2.4**Due: Wed, Feb. 28, 2007***Problem 1** (0 Points)

A2-1 in the FAQ to Homework 4 shows a visualization of two methods for constructing a pipeline. In method C, the interactive shell (denoted by `x[0]`) creates all child processes, but in method A, each new process creates the next process in the pipeline.

Suppose that the shell is required only to handle the case of only one pipeline and no `stdin/stdout` redirection; i.e., "`C[0] | C[1]`". Furthermore, suppose that the pipeline evaluation function has the following interface:

```
void eval_pipe( char **xargv[] );
```

where `xargv[i]` is a pointer to the i th pipeline command stored in `argv` format; i.e., `xargv[i][0]` is the command, `xargv[i][1]` is the first argument, `xargv[i][2]` is the second argument, etc. of the i th command.

- The diagram for method A indicates that the interactive shell `x[0]` waits for the first command `C[0]` to complete but not the other commands in the pipeline. Why doesn't the interactive shell in method A wait on all the child processes as in method C?
- The method A diagram shows `x[i]` creating pipe $i - 1$. Could `x[0]` create all pipes as in method C?
- Is it simpler for `x[0]` to create all pipes or for `x[i]` to create pipe $i - 1$?
- Draw a diagram of the file descriptor table for each of the processes executing commands `C[0]` and `C[1]`.
- Give the C/C++ code for evaluating the pipeline assuming that there is always one pipe and all commands are external commands.

Problem 2 (6 Points)

This problem considers pipelining methods A and C for an arbitrary number of pipes.

- Give the C/C++ code for the `eval_pipe` interface for evaluating an arbitrary length pipeline using method A. Assume that all commands are external commands and the wrappers for `Fork`, `Execvp`, and `Dup2` have been supplied.
- What are the properties of a correct execution of `eval_pipe`?

Problem 3 (0 Points)

The following is a sequence of commands (with line numbers) entered in rapid succession to the `bash` shell:

```
1  cat /etc/passwd | sleep 60 &
2  ps -o pid,ppid,pgrp,tpgid,cmd
3  ctrl-c
4  fg
5  ctrl-c
```

- What relationships will be shown by the output from line 2 regarding the process hierarchy, the process group(s), and the controlling terminal?
- What will be the effect of line 3?
- What will be the effect of line 4?
- What will be the effect of line 5?

Problem 4 (6 Points)

This problem considers the effect of `ctrl-c` in general and on various implementations of `xssh`.

- What is the difference between ignoring a signal and blocking a signal?
- We know that during the execution of `execvp`, Pending signals are cleared and all signal handling is reset to their default behavior. But it is possible that the user enters `ctrl-c` after entering a command but before the interactive shell has had a chance to call `execvp`. How should `xssh` ideally behave in this situation?
- What will happen if `xssh` just ignores `ctrl-c` (i.e., by calling `sigaction` with the `SIG_IGN` action) and the situation in Part b occurs?
- Explain what `xssh` must do to get the behavior in Part b. Code is not necessary.

Problem 5 (0 Points)

Consider the following parallel program:

```
int X; // Global (shared)
void inc() { int n; for (n=0; n<100; n++) if (X<10) ++X; }
void main() { X=0; parbegin inc(); inc() parend; print(X); }
```

The construct `parbegin S1; S2; ... parend` with statements `S1, S2, ...` means that the statements can execute in parallel subject to any synchronization primitives. In the above case, there is no synchronization between the two instances of the `inc` function.

Determine the smallest and largest value of the shared variable `X` that will be printed *and explain* how you arrived at this answer. Assume processes can execute at any relative speed and that a value can be incremented/decremented after it has been loaded into a register.

Problem 6 (0 Points)

This problem considers Peterson's 2-process algorithm given in class.

- a) Explain how the algorithm prevents one process from monopolizing the critical section; i.e., prevent starvation?
- b) Explain how the algorithm guarantees freedom from deadlock?
- a) What is the purpose of each of the semaphores X, Y, and Z[i]?
- b) What would be the effect of deleting the statement labeled Place A from the algorithm?

Problem 7 (6 Points)

Consider the following software-only mutual exclusion algorithm:

```

boolean blocked[2];
int who;
void P (int id) {
    while (TRUE) {
        blocked[id] = TRUE;
        while (who != id) {
            while (blocked[1-id]) who = id;
        }
        // --| critical section goes here |--
        blocked[id] = FALSE; // exit section
        // --| ... other processing ... |--
    }
}
void main () {
    blocked[0] = blocked[1] = FALSE;
    who = 0;
    parbegin ( P(0), P(1) );
}

```

- a) If the processes execute at the same rate as much as possible, in what order will the processes enter the critical section.
- b) Summarize how the algorithm attempts to guarantee mutual exclusion in the general case.
- c) The algorithm contains some flaws. Give an example where the algorithm exhibits starvation.
- d) Give an example where the algorithm exhibits livelock and explain why your example exhibits livelock.
- e) In what sense is the algorithm speed-sensitive?