Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder, Anuj Sehgal

Course: 320202
Date: 2010-04-14
Time: 10 min.

Quiz Sheet #5

Problem 5.1: *pipes and processes* (2+2+2 = 6 points)

Consider the following UNIX shell commands:

Command	Description
`find`	recursively descends the directory tree starting at the specified path and lists all files and directories on the standard output
`head`	displays the first few lines of the text received over standard input on the standard output
`tail`	displays the last few lines of the text received over standard input on the standard output
`less`	displays the entire text it receives over standard input on the standard output, allowing forward and backward navigation through the text; it does not quit until the user quits it manually
`echo`	writes the arguments it receives to the standard output
`ps`	lists the currently running processes

A user types the following commands a)-c), one-by-one, on the command line.

a) `find / | tail &`

b) `find / | head &`

c) `echo "hello world" | less &`

After issuing each of the commands a)-c), the user types `ps` to inspect the list of child processes of the shell. How many child processes will be listed after each of the commands a)-c)? Assume that there are no child processes before the user types one of the commands a)-c). Briefly explain your answer.

Solution:

a) The `find` process iterates through the whole file system and `tail` needs to wait until the last file has been listed. Since `find` takes time to complete, the user will see **two** processes running (excluding `ps`).

b) The `find` process again iterates through the file system, but `head` exits after the first few files have been listed. By the time the user has finished typing `ps`, both processes are done (since the pipe is closed early by `head` and this causes `find` to quit). Hence, the user will see **zero** processes running (excluding `ps`).

c) The `hello` process exists very quickly and will be done by the time the user has finished typing `ps`. However, `less` will continue to exist until `less` gets instructed to quit. Hence, the user will see **one** process running (excluding `ps`).

Problem 5.2: *signals* (6 points)

Consider the following C program:

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

static volatile sig_atomic_t wake_up = 0;

static void something(int dummy)
{
    wake_up = 1;
}

int main()
{
    sigset_t mask, old_mask;

    signal(SIGINT, something);
    sigemptyset(&mask);
    sigaddset(&mask, SIGINT);
    sigprocmask(SIG_BLOCK, &mask, &old_mask);

    while (!wake_up) {
        sigsuspend(&old_mask);
    }

    printf("Hello World!\n");
    return 0;
}
```

What happens when the program is executed? Explain.

Solution:

The program installs the function `something()` as a signal handler for the interrupt signal `SIGINT`. It then initializes the signal mask `mask` to an empty set and adds the signal `SIGINT` to it. The subsequent call to `sigprocmask()` adds the signals in the set `mask` to the set of signals to block (ignore). After returning from this call, the signal `SIGINT` is blocked and will not be delivered.

The program then enters a loop (assuming a `SIGINT` has not been received before the signal got blocked), suspending the execution with the original signal mask `old_mask`, where `SIGINT` was not blocked. The suspend loop continues until a `SIGINT` has been received and `wake_up` changes to 1. Afterwards, the program will print `Hello World!` before it exits.