

SYS2(OS) Week 4 Lab : Process Creation

Dr Poonam Yadav and Sven Signer

Process Information Commands

- “ps” command

- Provides a list of the system’s current processes
- Some useful options:
 - -e Show all processes
 - -F Show additional fields
 - -p pid Select a specific PID
- A comprehensive description can be found in the manual pages:
 - `man 1 ps`

- “pstree” command

- Shows system’s current processes as a pseudo-graphical tree structure
- Some useful options:
 - -p Show PIDs
 - -h Highlight the path to the current process
- A comprehensive description can be found in the manual pages:
 - `man 1 pstree`

Fork System Call

Description

- New processes are created with the fork system call.
- Duplicates the current process, creating a child process that is an (almost) identical copy of the parent process.
- Both processes continue executing from the fork point.
- Programs will typically immediately branch on the return value of the fork call so that the parent and child process can do different things.

Call Signature

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

```
pid_t fork(void);
```

- Return value differs between parent and child process:
 - Parent receives PID of the child.
 - Child process receives 0.
- pid_t is a type that stores a process ID, in practice it's just an alias to int.
- See manual page for further details:
man 2 fork

Wait System Call

Description

- Often some synchronisation between a parent and child process is required.
- This system call allows the parent process to wait for a child process to finish executing and (optionally) get some exit status information.
- Additional variants of the wait call allow the parent to check if the child has terminated without a blocking wait, or to wait on a specific child process.

Call Signature

```
#include <sys/wait.h>
```

```
pid_t wait(int *wstatus);
```

- Returns process id of terminated child process.
- If wstatus argument is not NULL, status information on the terminated child is stored into it.
- See manual page for further details:
`man 2 wait`

Fork Example

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

```
int main() {
    ...
```

```
    pid_t fpid = fork();
```

```
    if (fpid == 0) {
        printf("Child process\n")
        ...
        return 0;
    }
```

```
    else {
        printf("Parent process\n");
        ...
        wait(NULL);
        return 0;
    }
```

```
}
```

Fork point where the fork system call is invoked. Both parent and child process will continue executing from here.

Since we typically don't want the parent and child process doing the same thing, branch on the returned PID.

Parent process waits for child to terminate before exiting.

Exec Function

Description

- Allows a process to replace the program it is currently running.
- Often programs will want to start a new process running a different program, this is achieved by a fork after which the child immediately calls exec.

Call Signature

```
#include <unistd.h>

int execl(const char *path,
          const char *arg0,
          ..., (char *)0);
```

- Doesn't return (except on error).
- Path argument is a string containing the path to the new executable.
- Additional arguments are passed to the new program. The program name is given as the first argument.
- See manual page for further details: `man 3 exec`

Exec Example

```
#include <sys/wait.h>
#include <unistd.h>
```

```
int main() {
    pid_t fpid = fork();
```

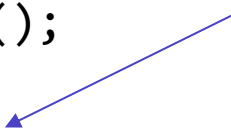
```
    if (fpid == 0) {
        execl("/bin/echo", "echo", "Hello World", NULL);
```

```
    }
    else {
        wait(NULL);
    }
```

```
    return 0;
```

```
}
```

Child process replaces its program with the echo program. The echo program simply prints its arguments (here "Hello World") to the terminal.



Utility Functions

```
#include <stdlib.h>
```

```
void exit(int status);
```

- Terminate the process with the given exit status.

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

```
unsigned int sleep(unsigned int seconds);
```

```
unsigned int usleep(unsigned int microseconds);
```

- Wait for the given number of seconds/microseconds.
- Returns 0 on success.

Assignment



This lab session looks at understanding the behaviour of new processes created with the fork system call.

An example program `simple_fork.c` is provided on the VLE to get you started.