CSCB09: Software Tools and Systems Programming

Bianca Schroeder
bianca@cs.toronto.edu
IC 460

The plan for today

- Processes
  - How to create new processes …

- Why would you want to have a program that creates new processes?

Creating new processes with fork

- Why would you want to have a program that creates new processes?

- Motivation: Hardware is not getting that much faster anymore

  => use more hardware to solve a problem

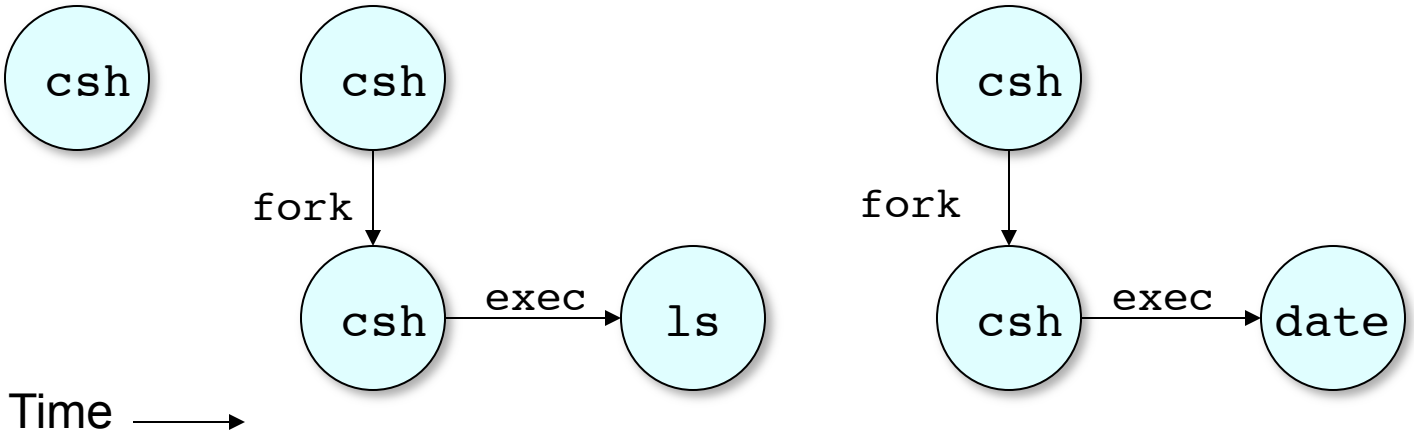
  - Multiple processors/cores
  - Multiple machines

  => Need to write a parallel program with multiple processes to exploit parallelism in hardware

Examples: Parallel programs

- Example: Web search engine
  - Web indices store counts of word occurrences for different files (web pages)
  - A search looks up a word in many different indices in parallel
- Example: A web server
  - Multiple processes serving web requests in parallel
- Example: A unix shell

Example: A shell



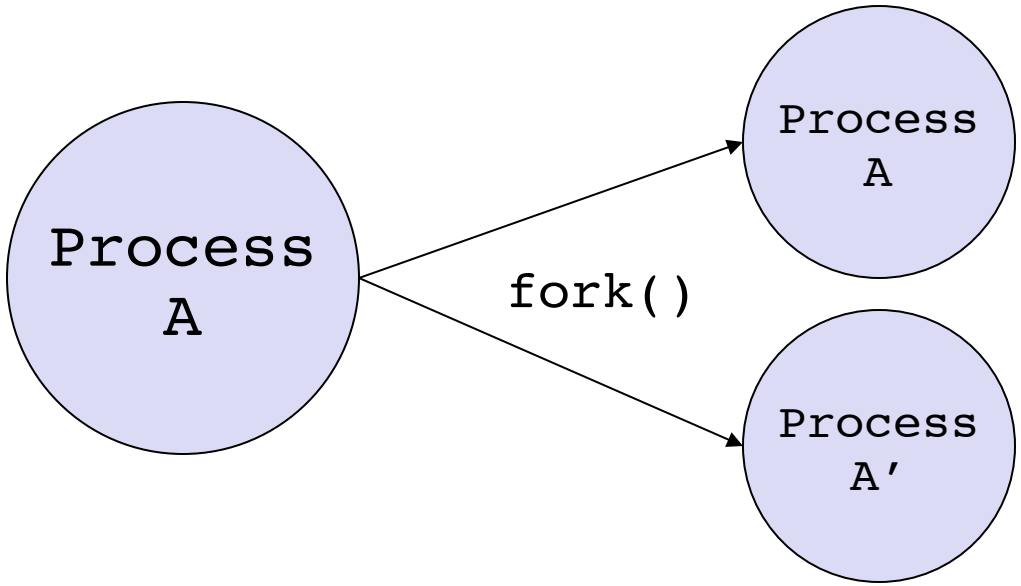
- When a command is typed, shell forks and then execs the typed command.

Processes can create other processes

- The fork system call creates a child process:

```
#include <unistd.h>
pid_t fork(void);
```

- Fork creates a **duplicate** of the currently running program.
- Both processes run concurrently and independently.
- After fork() both execute the next instruction after fork.



What is different between parent and forked child

- The child gets a new PID (process ID) and PPID
- The return value from the fork call is different:
  - –On success
    - fork() returns 0 to the child
    - fork() returns the child's PID to the parent.
  - –On failure (no child created) fork returns -1 to parent

Fork example

```
int main ()
{
   pid_t pid;
   pid = fork();
   if (pid < 0) {
      perror("fork()");
   } else if (pid > 0) {
      printf("parent\n");
   } else {  /* pid == 0 */
      printf("child\n");
   }
   return 0;
}
```

Another fork example

Original process (parent)

```
int i; pid_t pid;
i = 5;
printf("%d\n", i);
/* prints 5 */
pid = fork();
/* pid == 677 */
if (pid > 0)
    i = 6;
printf("%d\n", i);
```

Child process

```
int i; pid_t pid;
i = 5;
printf("%d\n", i);

pid = fork();
/* pid == 0 */
if (pid > 0)
    i = 6;
printf("%d\n", i);
```

- Each process gets their own copy of the address space
- After call to fork, variables are not shared.

Let’s take a look at the first work sheet!

When does a child terminate?

- Like any other program:
  - The program's main function returns
  - The program calls `exit`
    - `void exit(int status);`
    - (only the program calling exit will exit; not its children)
  - The program receives a signal that causes it to terminate
    - Will see next lecture what that means …
- Programs have an exit status
  - Exit status is set by call to `exit` or main's `return`
  - Exit status of most recent command stored in `$?` in most shells

`wait()`

- A parent might want to wait for a child to complete
- A parent might want to know the exit code of a child

- System call to wait for a child
  - `pid_t wait(int *status)`

- `wait` suspends execution of the calling process until one of its children terminates

wait()

- System call to wait for a child
  - `pid_t wait(int *status)`
- After calling wait() a process will:
  - block if all of its children are still running
  - return immediately with the PID of a terminated child, if there is a terminated child
  - return immediately with an error (-1) if it doesn't have any child processes.

Info returned by `wait()`

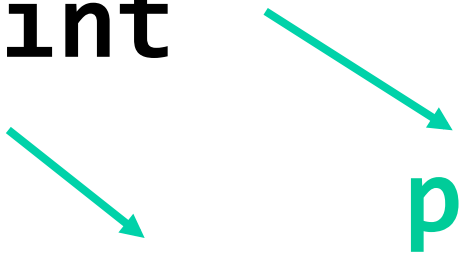
- System call to wait for a child
  - `pid_t wait(int *status)`
- Returns the pid of the terminated child or -1 on error
- `status` encodes the exit status of the child and how a child exited (normally or killed by signal)
- There are macros to process exit status:
  - `WIFEXITED` applied to `status` tells you if child terminated normally (i.e. by calling exit or return)
  - `WEXITSTATUS` gives you the exit status

Example using `wait()`

```
int main (void) {
    pid_t child;
    int status, exit_status;

    if ((child = fork()) == 0) {
      sleep (5);
      exit (8);
    }
    wait (&status);
    if (WIFEXITED(status)) {
     exit_status = WEXITSTATUS(status);
     printf ("Child %d done: %d\n", child, exit_status);
    }
    return 0;
}
```

treat as
an **int**



```
pid_t fork()
pid_t wait(int *status)
void exit(int status)

WIFEXITED(status)
WEXITSTATUS(status)
```