More on Processes

Ref: Chapter 5 of [HGS].

Additional Program Examples:

- A child process that prints its arguments (Handout 15.1).
- A simple version of the `system` library function of Unix (Handout 15.2).

Shared File Descriptors:

- Recall: Parent and child share file descriptors.
- Changes made to the file by the parent are visible to the child and vice versa.
- **Example:** Handout 15.3.

System call `fcntl`:

```
int  fcntl(int fd, int cmd, ...);
```

- Performs a variety of control functions associated with open files.

- Headers: `<sys/types.h>`, `<unistd.h>` and `<fcntl.h>`.
- `fd`: File descriptor of an open file.
- `cmd`: Specifies the function to be performed.
- The number of parameters depends on `cmd`. (Some calls may have just two parameters.)
- Returns `-1` if there is an error.

Example 1:

- The value `F_GETFL` for `cmd` can be used to obtain the current file status flags.
- The value returned by `fcntl` must be bitwise anded with `O_ACCMODE` to obtain the status flags.

Sample code segment:

```
if ((x = fcntl(fd, F_GETFL)) != -1) {
  if ((x & O_ACCMODE) == O_RDWR)
    printf("Status: read-write.\n");
}
```

Example 2:

- The value F_SETFL for cmd can be used to change the current file status flags.
- Some changes may not be permitted.

Sample code segment:

```
if (fcntl(fd, F_SETFL, O_APPEND)
     == -1) {
  fprintf(stderr, "Failure.\n");
}
```

Example 3:

- The value F_SETFD for cmd can be used to turn on (or off) the "close-on-exec" (COE) flag for a file.
- If this flag is on, then the file is closed when any member of the exec family is invoked.

Sample code segment:

```
if ((fd = open("d.dat", O_RONLY))
    == -1) {
  fprintf(stderr, "Error.\n");
  exit(1);
}
if (fcntl(fd, F_SETFD, 1) != -1) {
  printf("COE flag is on.\n");
}
```

System call exit:

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

- Terminates a process.
- Header: <stdlib.h>,
- status: Value of exit status to be given to parent.
- Convention: Exit status = 0 for a normal exit and non-zero for an error exit.

- Some actions performed by `exit`:
  - Invokes other functions (exit handlers) registered using the `atexit` routine.
  - Closes all open files.
  - Restarts a waiting parent.

Library function `atexit`:

```
int  atexit(void (*f)(void));
```

- Header: `<stdlib.h>`,
- The parameter is a function pointer.
- "Registers" the specified function `f` as an exit handler.
- Returns `-1` if the specified function can't be registered. (However, `errno` is not set.)
- The function to be registered cannot have any parameters.
- The same function may be registered multiple times. (Generally, a total of up to 32 items can be registered.)

- Functions are invoked in an order which is the reverse of the registration order.

Program example: Handout 15.4.

A Simple Shell:

- Simpler version of the program discussed in Section 5.9 of [HGS].
- Shows the main part of the shell.
- Program repeats the following steps:
  (a) Produce prompt.
  (b) Read command.
  (c) Parse command to obtain parameters.
  (d) Fork a child process and use exec to execute the command.
  (e) Wait for the child to exit.
- Program exits when the user types CTRL-D.
- Doesn't support background processes, I/O redirection, etc.
- Details: Handout 15.5.