

# CSE2031 Software Tools - System Calls, Processes

Summer 2010

Przemyslaw Pawluk

Department of Computer Science and Engineering  
York University  
Toronto

June 22, 2010

## 1 Files - review

## 2 Processes

- Low-level process creation
- Control of process

## 3 Filters

CSE2031

Software

Tools -

System Calls,

Processes

Przemyslaw

Pawluk

Files - review

Processes

Low-level

process

creation

Control of

process

Filters

## 1 Files - review

## 2 Processes

- Low-level process creation
- Control of process

## 3 Filters

## Methods

- `fopen` – opens a file and returns a pointer to `FILE` structure
- `fclose` – closes a file (also writes a buffer content if any)
- `fflush` – writes a buffer into a file
- `read`
  - `getc` – reads one char from the input file
  - `fscanf` – reads input from file like `scanf`
- `write`
  - `putc` – prints a char into file (buffered)
  - `fprintf` – prints a formatted string into a file

## Methods

- `fopen` – opens a file and returns file descriptor
- `create` – closes a file (also writes a buffer content if any)
- `read` – reads  $n$  bytes form file into a buffer
- `write` – writes  $n$  bytes form buffer into a file

## 1 Files - review

## 2 Processes

- Low-level process creation
- Control of process

## 3 Filters

# Low-level process creation

CSE2031

Software

Tools -

System Calls,  
Processes

Przemyslaw  
Pawluk

Files - review

Processes

Low-level  
process  
creation

Control of  
process

Filters

## How to call a program from another program?

C allows us to call a program from our code (without returning) by two commands `exec1p` and `execvp`.

```
1  execvp(PATH, PROGNAME, ARGS ...);
```

- PATH is a path containing a program name
- PROGNAME is a first element of the argv array
- ARGS are subsequent command line arguments where the last one is NULL (0)

```
1  execvp("date", "date", (char *) 0);
```



Works exactly the same way, however accepts a array or arguments, so you do not need to know a number of arguments in advance.

# How it works?

CSE2031

Software

Tools -

System Calls,  
ProcessesPrzemyslaw  
Pawluk

Files - review

Processes

Low-level  
process  
creationControl of  
process

Filters

```
1  int main(int argc, char * argv[]){  
2      execlp("echo", "echo", argv[1]);  
3      error("cannot execute echo %s", argv[1]);  
4  }
```

## Execute and get the control back

fork allows us to call a program and regain control after running a program with `execlp/execvp`

## How to use it?

```
1  int procid = fork();
2  if (procid == -1) {
3      error("cannot_create_child_process");
4      exit(-1);
5  }
6  else if (procid == 0) { /* child process */
7      execlp("data", "data", (char*)0);
8      error("cannot_execute_data");
9  } else { /* Parent */
10     /* Parent can do something or wait for a child */
11     wait(&status);
12 }
```

1

```
wait(&status);
```

- wait makes parent to wait for a result from child
- status encodes eight bits (low-order) an exit status of child where 0 mean normal termination and non-zero some kind of error

## This is not covered by any of our textbooks!

The signals are defined in the include file `<signal.h>`.

**SIGABRT** – Abnormal termination, such as instigated by the abort function. (Abort.)

**SIGFPE** – Erroneous arithmetic operation, such as divide by 0 or overflow. (Floating point exception.)

**SIGILL** – An invalid object program has been detected. This usually means that there is an illegal instruction in the program. (Illegal instruction.)

**SIGINT** – Interactive attention signal; on interactive systems this is usually generated by typing some break-in key at the terminal. (Interrupt.)

**SIGSEGV** – Invalid storage access; most frequently caused by attempting to store some value in an object pointed to by a bad pointer. (Segment violation.)

**SIGTERM** – Termination request made to the program. (Terminate.)

# Send and receive signals

CSE2031

Software

Tools -

System Calls,  
ProcessesPrzemyslaw  
Pawluk

Files - review

Processes

Low-level  
process  
creationControl of  
process

Filters

## Receive

```
void (*signal (int sig, void (*func)(int)))(int);
```

## Send

```
int raise (int sig);
```

CSE2031

Software

Tools -

System Calls,  
Processes

Przemyslaw  
Pawluk

Files - review

Processes

Low-level  
process  
creation

Control of  
process

Filters

## 1 Files - review

## 2 Processes

- Low-level process creation
- Control of process

## 3 Filters

# What is a filter in Unix?

CSE2031

Software

Tools -

System Calls,  
Processes

Przemyslaw  
Pawluk

Files - review

Processes

Low-level  
process  
creation

Control of  
process

Filters

Filter is a program that has following properties:

- Read text input line by line (from stdin by default)
- Perform some transformation
- Write some output (to stdout by default)



# What can we do with filters?

CSE2031

Software

Tools -

System Calls,  
ProcessesPrzemyslaw  
Pawluk

Files - review

Processes

Low-level  
process  
creationControl of  
process

Filters

- Filters are very common tools in Unix-like systems. Many standard commands are actually filters (grep, cut etc.).
- Filters can work together as parts of pipes

```
grep pawluk marks.txt | cut -f4
```

# How to write a filter in C

CSE2031  
Software  
Tools -  
System Calls,  
Processes

Przemyslaw  
Pawluk

Files - review

Processes

Low-level  
process  
creation

Control of  
process

Filters

Your program should do following things:

- Process the stdin line by line
- Do some transformations based on the input read
- Write output to the stdout
- Write any errors into stderr

## Reverse

Let's write a filter that reverses a word in the stdin and writes result to the stdout. We will call it reverse.