

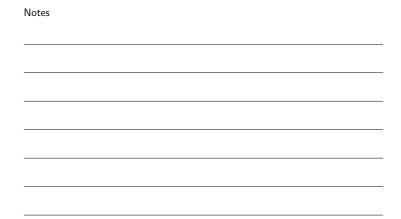
## CSE2031 Software Tools - System Calls, **Processes**

Summer 2010

Przemyslaw Pawluk

Department of Computer Science and Engineering York University Toronto

June 22, 2010





## Table of contents

1 Files - review



Processes

- Low-level process creation
- Control of process

Filters



Notes

Notes





#### High-level access

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
  - $\bullet\,$  getc reads one char from the input file
  - fscanf reads input from file like scanf
- - putc prints a char into file (buffered)
    fprintf prints a formatted string into a file



## Low-level access

CSE2031 Software Tools -System Calls, Processes

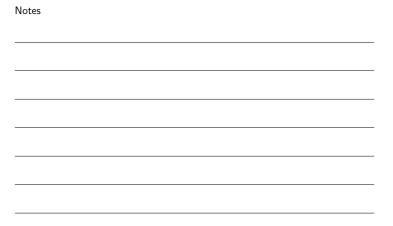
Processe Przemysl

Files review

Low-level process creation Control of process Filters 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
- $\bullet$  write writes n bytes form buffer into a file

5 / 19





#### Low-level process creation

Software Tools -System Calls, Processe

Pawluk

Processes

Low-level process creation

How to call a program from another program?

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

Notes			



# execlp

CSE2031 Software Tools -System Calls, Processes Przemyslav

execlp(PATH, PROGNAME, ARGS ...);

- Pawluk
- Files review

Low-level process creation Control of process

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

execlp("date", "date", (**char** \*) 0);

Notes		



#### execvp

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

Notes			



# How it works?

```
int main(int argc, char * argv[]){
   execlp("echo", "echo", argv[1]);
   error("cannot_execute_echo_%s", argv[1]);
```

Notes			



# fork

Execute and get the control back

a program with execlp/execvp

How to use it?	1	
<pre>int procid = fork();</pre>		
<b>if</b> ( procid==-1){		
error("cannot_create_child_process");		
exit(-1);		
}		
<pre>else if(procid==0){ /* child process */</pre>		
execlp("data", "data", $(char*)0$ );		
error("cannot_execute_data");		
} else{ /* Parent */		
/* Parent can do something or wait for a ch	ild	*,
wait(&status);		
l		

 ${\tt fork} \ {\tt allows} \ {\tt us} \ {\tt to} \ {\tt call} \ {\tt a} \ {\tt program} \ {\tt and} \ {\tt regain} \ {\tt control} \ {\tt after} \ {\tt running}$ 

Notes			



#### wait and status

wait(&status);

- $\bullet$  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

Notes			



#### Signals

#### 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.)

Notes

Notes



#### Send and receive signals

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

Send

int raise (int sig);



# What is a filter in Unix?

ter 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)	

Notes



## What can we do with filters?

grep pawluk marks.txt | cut -f4

• Filters can work together as parts of pipes

• Filters are very common tools in Unix-like systems. Many standard commands are actually filters (grep, cut etc.).

Notes			



## How to write a filter in ${\sf C}$

Your program should do following things:

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

Notes		



# Example

CSE2031 Software Tools -System Calls,

Przemys

Files review

Processes
Low-level
process
creation
Control of
process

Reverse

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

Notes			
Notes			
Notes			