**CS241 Lecture 6 September 4, 2015 - Angrave and Karrels - Processes**

How are text files different on Unix vs. Windows?

How do I see the list of current processes?

What’s this “bash” thing?

How do I start a program from within my program?

What’s funny about the arguments to execl()?

Where did my program go?

How can I start a program and leave mine running too?

How can I tell what error execl() ran into?

When I run “ps”, the shell runs “/bin/ps”. How does it know where to find the program?

Where are environmental variables stored?

How do they get set for a new process?

How can I access them from C?

What is “where” and how could I implement it?

**Demo 1 – Run another program**

#include <unistd.h>

#include <stdio.h>

#include <errno.h>

int main() {

printf("Starting ps command...\n\n");

// int result = execl("/bin/ps", "ps", "-F", (char\*)NULL);

int result = execl("ps", "ps", "-F", (char\*)NULL);

if (result == -1) {

perror("trying to run ps command");

} else {

printf("\nps command finished\n");

}

return 0;

}

**Demo 2 – fork()**

#include <stdio.h>

#include <unistd.h>

int main() {

printf("About to fork up this demo...\n");

fork();

printf("It's alive!!!\n");

return 0;

}

**Demo 3 – fork() and pids**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

printf("About to fork up this demo...\n");

pid\_t pid = fork();

printf("fork returned %d\n", (int)pid);

return 0;

}

**Demo 4 – Parent and child pids**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

printf("About to fork up this demo...\n");

pid\_t pid = fork();

pid\_t mypid = getpid();

printf("[%d] fork() returned %d\n", (int)mypid, (int)pid);

return 0;

}

**Demo 5 – fork() / exec() / wait()**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

printf("Running \"ps\" the hard way...\n\n");

pid\_t pid = fork();

if (pid == 0) {

// child process

sleep(3);

int result = execl("/bin/ps", "ps", "-F", (char\*)NULL);

if (result == -1) {

perror("trying to run ps command");

}

} else {

// parent process

printf("[parent] Waiting for child to finish...\n");

int status;

waitpid(pid, &status, 0);

printf("[parent] Child process returned %d\n", status);

}

return 0;

}

**Demo 6 – environ / getenv() / setenv() / putenv()**

// either one of these enables setenv()

// #define \_POSIX\_C\_SOURCE 200112L

#define \_BSD\_SOURCE

// either one of these enable putenv()

// #define \_SVID\_SOURCE

#define \_XOPEN\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

extern char \*\*environ;

int main() {

char \*\*vars = environ;

int i=0;

while (\*vars) {

printf("[%d] %s\n", i++, \*vars);

vars++;

}

putchar('\n');

printf("HOME = %s\n", getenv("HOME"));

printf("PATH = %s\n", getenv("PATH"));

printf("FOOBAR = %s\n", getenv("FOOBAR"));

putchar('\n');

// setenv() saves a copy of the strings you give it printf("set FOOBAR to baz\n");

if (setenv("FOOBAR", "baz", 0))

printf("Error calling setenv\n");

putchar('\n');

printf("Checking FOOBAR: %s\n", getenv("FOOBAR"));

putenv("FOOBAR=putenv works too");

printf("Checking FOOBAR: %s\n", getenv("FOOBAR"));

// putenv() saves the pointer you give it, so you can change the data

char myvar[] = "MYVAR=hello there";

putenv(myvar);

printf("set MYVAR to %s\n", getenv("MYVAR"));

strcpy(myvar, "MYVAR=goodbye!");

printf("MYVAR is now %s\n", getenv("MYVAR"));

return 0;

}