CS 214 / 2022-05-02

===================

open(path\_name, O\_RDONLY);

open(path\_name, O\_WRONLY|O\_TRUNC|O\_CREAT, mode);

other ways to create file descriptors:

socket(), accept(), pipe(), dup(), dup2()

the OS keeps track of all open files

file descriptors are a many-to-one relation with open files

fork(), wait(), and exec()

fork duplicates the current process

-> the only difference will be the value that fork() returns

in the new process (child) fork returns 0

in the old process (parent) fork returns the child's PID

the parent is responsible for cleaning up the child by calling wait()

wait will block the parent until the child terminates and

provide information about how the child terminated

if the child terminates before the parent calls wait, it becomes a zombie

if the parent terminates without calling wait, the child becomes a orphan

-> typically, the OS will "adopt" orphans and wait for them

if an orphan process terminates, it becomes a "zombie orphan"

exec() changes which program the process is executing

-> the new program starts from the beginning

-> some process-level information preserved (like open files)

-> otherwise, the old program is lost

-> no way to resume

we can use fork() wait() and exec() to start a program and wait for it

to finish

multithreading

-> what if we ran multiple processes (threads) in the same shared memory?

pthread\_create() and pthread\_join()

condition variables

-> allow threads to wait for a signal from another thread

barriers

-> allow multiple threads to wait until a certain number of threads

are waiting

semaphores

-> generalization of mutex and condition variables

deadlock

-> mutual exclusion

-> hold and wait

-> no preemption

-> circular wait

suspends current process until a child process ends

pid\_t wait(int \*wstatus);

wait() returns the PID of the child process that ended

if wstatus != NULL, it writes the exit status of the child into wstatus

pid\_t child, child2;

int wstatus;

child = fork();

if (child == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (child == 0) {

execl(subprogram, subprogram, arg1, NULL);

perror(subprogram);

exit(EXIT\_FAILURE);

}

child2 = wait(&wstatus);

// we would normally expect child == child2 here, since we didn't start any other children

if (wstatus != 0) {

// something went wrong with our subprogram

}