

Project 5 - User programs II

PintOS

In this project you are asked to implement two system calls: *wait* and *exec*. To implement *exec*, you need synchronization and also you need to check the pointers passed by the user which requires some understanding of memory paging.

The *wait()* system call

The calling process blocks until its child *p*, which has id *pid*, has finished. Here is the syscalls declaration:

- *int wait (pid_t pid)*

The return value must be:

- the exit status of *p*, if all was fine (even if *p* finished before *wait* was called!);
- -1 if *p* was killed;
- -1 if *p* is not a direct child of the caller (this makes it easier);
- -1 if the calling process already waited for that *pid* before.

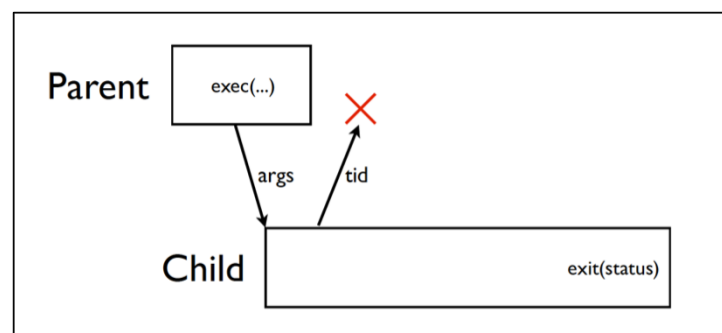
The *exec()* system call

A user program can use this system call to execute a command by creating a child process. Here is the syscall's declaration:

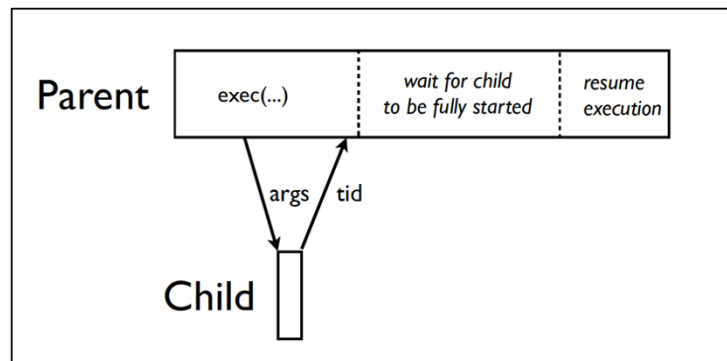
- *pid_t exec (const char *cmd_line)*

It tries to run the executable given in *cmd_line*. The return value is the *pid* (tid) of the child process (or -1 in case of error). A correct implementation synchronize between parent and child processes in order to prevent the following corner cases:

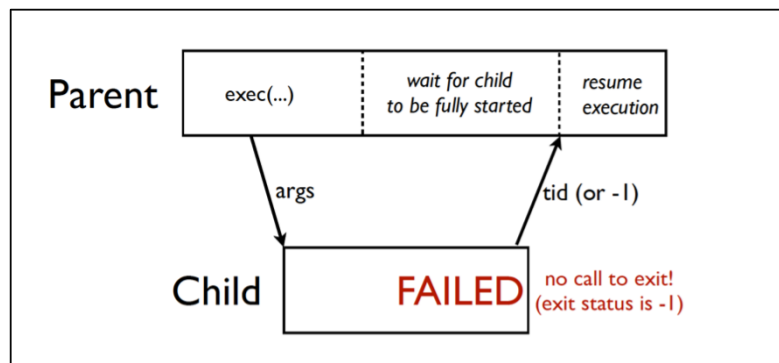
The parent thread must wait for the child creation:



The child thread may finish too fast:



The child thread may fail before telling some status:



Hints

The synchronization between parent and child process can be handled with `thread_block()`, or just a semaphore.

Pintos semaphore API

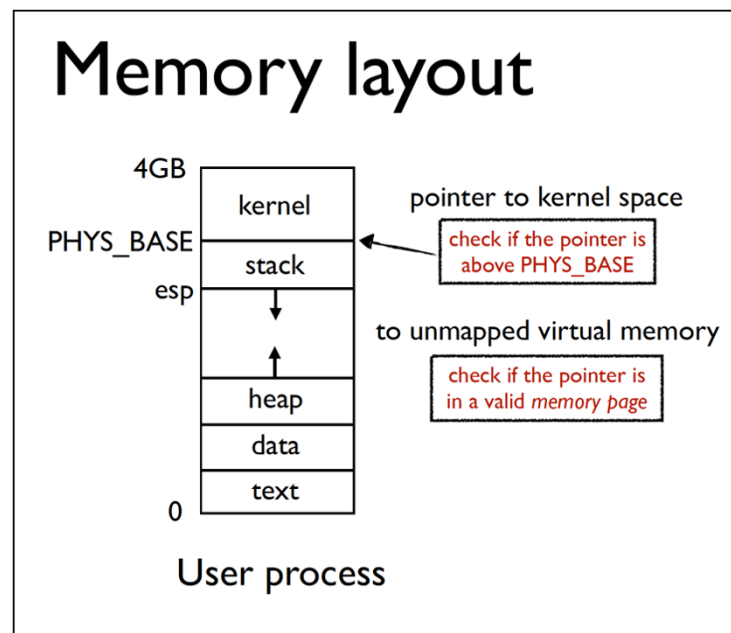
Pintos semaphore is a tool for synchronizing threads. It can solve the “wait for child” problem of the `exec` syscall. The API is available in “`thread/synch.h`”.

- **struct semaphore** semaphore type; must be initialized with `sema_init`;
- **sema_init** (**struct semaphore** * *s*, **unsigned val**) initialize semaphore pointed by *s* with value *val*;
- **sema_down** (**struct semaphore** * *s*) if *s* is 0 (zero), block the calling thread and put it in a list waiting for *s*; otherwise, decrement *s* by 1;
- **sema_up** (**struct semaphore** * *s*) if *s* is 0 (zero) and there is some thread waiting for *s*, unblock one of the threads that are waiting for *s*; otherwise, increment *s* by 1;
- neither **sema_up** or **sema_down** can be interrupted (they’re atomic)

Memory Access

The user may provide an invalid pointer in a syscall. It is important to check the pointer before using it. You should check if the pointer is:

- a null pointer;
- a pointer to kernel address space;
- a pointer to unmapped virtual memory; make sure user pointer is in a valid memory page (hints in **vaddr.h** and **pagedir.c**).



Tests

Your implementation should pass the tests:

- *exec-once*
- *exec-arg*
- *exec-multiple*
- *exec-missing*
- *exec-bad-ptr*
- *wait-simple*
- *wait-twice*
- *wait-bad-pid*

Readings

- PintOS documentation
 - Chapter 3 (Specially sections 3.1.4 and 3.3.4.)