Università della Svizzera italiana

Faculty of Informatics

Operating Systems

# PintOS: Report Project 3

Group 11
Machado Ferreira, Zeneli Jora, Del Pozzo Daniele

Spring Semester, April 17, 2025

## 1 Questions

Consider file `thread.h`

1. Which attributes did you add to `struct thread`?
2. What is the purpose of each one of them?

Consider file `thread.c`

1. We said that the child should keep track of the parent. Is it the parent or the child that takes care of this? Inside which function?
2. For the userprog tests, the thread name must be just the filename that it executes, excluding any arguments passed along with it in the command line. In which function of `thread.c` do you take care of this?

Consider file `process.c`

1. Are arguments to be pushed on the stack before or after loading the program?
2. Once arguments are pushed on the stack, are you padding the memory to keep the 32-bit alignment of Pintos addresses? If yes, inside which function?
3. Tests require the exit status of a process to be printed as follows:

   ```
   printf("%s: exit(%d)\n", <program-name>, <exit-status>);
   ```

   Inside which function do you do it?
4. Inside which function and at which line of code do you unblock the parent of a child? Is it the parent or the child doing this?

Consider file `pintos/userprog/syscall.c`

1. Which of the macros in `pintos/lib/user/syscall.c` corresponds to the `exit` syscall? Which one corresponds to the `write` syscall?
2. Consider `pintos/lib/user/syscall.c`, what is the code of the interrupt generated by the macros?
3. What is the number associated to the `exit` syscall? And to the `write` syscall?

# 2 Answers

File `thread.h`

1. The following attributes were added to `struct thread` section:
   - `tid_t pid;`
   - `bool beingWaited;`
   - `bool hasExited;`
   - `int proc_status;`
2. 
   - `pid`: Stores parent's thread ID.
   - `beingWaited`: Stores a boolean flag indicating whether the thread is being waited on by its parent. Used to check whether any thread should be unblocked when the current thread terminates.
   - `hasExited`: Stores a boolean flag indicating whether the thread has finished execution. This is used by the parent to check if the child has already exited, giving the parent the go ahead to retrieve exit status, avoiding unnecessary blocking.
   - `proc_status`: Stores the exit status of the process. The attribute is filled when a thread exits and the exit code is stored in this attribute. This value is then returned to the waiting parent if there is one.

File `thread.c`

1. It is the child that keeps track of its parent using the `pid` field of `threads`. This is handled inside the function `thread_create`. When a new thread is created, it assigns the parent's thread ID to the `pid` field of the child:

```
#ifdef USERPROG
  t->pid = thread_current()->tid;
```

2. The function `thread_create()` takes as parameters the name and arguments of the functions as `char` that are already distinct. This way when the name is passed to the `init_thread()`, the thread name will be set correctly, without any arguments passed along in the command line.

File `process.c`

1. Arguments are pushed on the stack after the program has been loaded. This can be seen in the order of execution in the `start_process()` function, as a call to `load()` will happen before the arguments are pushed to the stack:

```
success = load (exec_name, &if_.eip, &if_.esp);
...
for (token = strtok_r(file_name_, " ", &save_ptr);
     token != NULL; ...)
```

Any mishap when loading will be caught by the `if (!success)` statement that is executed between loading and pushing to stack.

2. Yes, memory is padded to maintain 32-bit alignment of addresses. This is done inside `start_process()` by calling the `align_stack()` function:

   ```
   sp = align_stack(sp);
   ```

   The helper function `align_stack()` ensures the stack pointer is aligned to a multiple of 4 bytes.

3. This is handled in `process_exit()` in `process.c`:

   ```
   printf("%s: exit(%d)\n", curr->name, curr->proc_status);
   ```

   As can be seen that the printf statement follows the requirements set by the tests

4. The child is responsible for unblocking its parent. This is done in the `process_exit()` function at the lines:

   ```
   if (curr->beingWaited) {
     struct thread * pt = thread_get_by_tid (curr->pid);
     if (pt != NULL) {
       thread_unblock(pt);
     }
   }
   ```

   The call to `thread_unblock(pt)` where pt is the parent thread, is responsible for unblocking the parent. This happens if and only if there is a parent thread waiting on the child. This is enforced by the `if` statement that contains the `thread_unblock(pt)` call.

File `pintos/userprog/syscall.c`

1. The macros that correspond to the `exit` and `write` system calls are:
   - `SYS_EXIT` –> `exit` system call.
   - `SYS_WRITE` –> `write` system call.
2. The interrupt number used to invoke system calls is `0x30`, as can be seen during system call initialization:

   ```
   intr_register_int (0x30, 3, INTR_ON, syscall_handler, "syscall");
   ```

3. The numeric codes for syscalls are:
   - `SYS_EXIT` is associated with syscall number **1**.
   - `SYS_WRITE` is associated with syscall number **4**.