SUBMISSION .

“Lab Report #6.”

CSE233

Sherif M. Haredy (I.D.: 223107334; MAJOR: C.S.)
GALALA UNIVERISTY, EGYPT

Prof. Amr Hefny

—

November 18, 2025

I Implementation

A struct `job` handles the process related data:

- PID number
- wait status

- interrupt code
- exit code

The handler routine `sigchld_handler` works by checking whether the current process either equals `child2`'s PID or not:

```
if (proc.p != child2.p)
```

If it does not, it updates wait status and interrupt code; if it does, then it calls `_exit(0)`, which terminates the parent if an interrupt signal is made.

The signal is sent via `kill()` according to the process' exit code.

The main routine forks two processes, `child1` and `child2` (prototyped globally), and runs:

```
while (1) { }
```

II SIG Reader

This routine is entirely ancillary and was not required per spec sheet.

`rint.c` implements a small pipe that allows reading each `job` from each process, which saves the interrupt code; this allows inspection of how each process was interrupted.

A subroutine forks a `_read` process which calls `sig_read()`, feeding the interrupt code for the given child process.

III Testing

The `script/` folder contains a shell script which runs a series of checks that match the lab's spec sheet. A CI runner works through it in order to verify success.

The errors depend on the value of `process.interrupt` and whether:

```c
printf("Child1␣PID:␣%d␣Parent␣PID:␣%d\n\n", getpid(), getppid());
```

messages are successful and correct, in order and format.

References

- Source Code: `https://github.com/sherif6931/oslab6`
- Lab Spec: c.f. `README.md` in the repository

This LaTeX doc's text is a copy of the `dev` *to* `main` *successful PR.*