📋 Homework-3: Process Controller

> ⚠️ **Important:** This assignment focuses on basic process creation using fork(), exec(), signal handling, and collecting child exit status. Students may adapt examples from instructor provided coding examples.
>
> ⚠️ **There is only one submission - RESUBMISSION IS NOT ALLOWED. You must test your implementation with the provided testing framework on the Ocelot server before the final submission.**

📧 Grader Contact Information

For any questions or concerns related to assignment grading, please reach out to the TA listed below.

Communication Guidelines:

- Ensure that all communication is polite and respectful
- **You must contact the Graders first for any grading issues**
- If the matter remains unresolved, feel free to reach out to the instructor

Name: Instructor — bbhatkal@fiu.edu (⚠️ via the Canvas inbox only)
Section: COP 4610-U01 (84957)

Name: Angie Martinez — amart1070@fiu.edu
Section: COP 4610-U02 (85031)

Name: Srushti Visweswaraiah — svisw003@fiu.edu
Sections: COP 4610-UHA (85236)

☑️ Section 1: Homework Overview

To create a process controller that executes commands sequentially using child processes and reports their exit status.

☑️ Section 2: Learning Objectives

- Create processes using the `fork()` system call.
- Load and execute a new program in the child process using `execvp()`.
- Implement basic signal handling for timeout management and graceful termination.
- Collect and report child exit status using `waitpid()`.

- Understand the process lifecycle and interpret exit codes.

☑ Section 3: Problem Description

You will implement a **process controller** that executes commands **sequentially** using child processes. For each command, the controller will:

- Create a child process.
- Execute the command in the child using `execvp()`.
- Wait for the child process to complete (successfully or unsuccessfully) using `waitpid()`.
- Report the child's exit status to the user.

This implementation requires adapting and integrating concepts from multiple practice code examples provided by the instructor.

Key Features

1. **Sequential Execution** – Execute one command at a time (no parallel/concurrent execution).
2. **Process Creation** – Use `fork()` to create child processes.
3. **Command Execution** – Use `execvp()` to run commands in the child process.
4. **Signal Handling** – Use `SIGALRM` to handle timeouts and ensure graceful termination.
5. **Exit Status Reporting** – Collect and display child exit codes using `waitpid()`.
6. **Error Handling** – Handle invalid commands and enforce timeout rules appropriately.

☑ Section 4: Provided Files Framework

📖 **Refer to these files** for understanding the exact requirements.

⚠️ **Please do not make any modifications to the framework files provided.** You are required to implement **only** the file `submission_HW3.c`. Any changes to the framework files may result in errors during evaluation or loss of credit.

Folder Structure

```
PROVIDED_FILES/
├── submission_HW3.c          # Template file (implement your solution here)
├── process_controller.h      # Header file (Do not modify!)
├── driver.c                  # Driver code (Do not modify!)
├── Makefile                  # Makefile (Do not modify!)
├── autograder_HW3.sh         # Testing script (Do not modify!)
└── Testing/                  # Testcases (Do not modify!)
    ├── Testcases/
    |   ├── input1.txt        # Valid commands only
    |   ├── input2.txt        # Invalid commands only
    |   └── input3.txt        # Mixed valid/invalid commands
    └── Expected_Output/
        ├── output1.txt       # Expected output for test 1
        ├── output2.txt       # Expected output for test 2
```

```
└── output3.txt      # Expected output for test 3
```

☑ Section 5: Developing and Testing Your Implementation

> ⚠ **DO NOT modify the following provided framework files:**
> autograder_HW3.sh, batchgrader_HW3.sh, process_controller.h, driver.c, Makefile, and the testcases.

Required Function

You must implement this function in `submission_HW3.c`:

`void execute_command(char *args[])`

- **Input**: NULL-terminated array of command and arguments
  - Example: `char *args[] = {"echo", "hello", NULL};`
- **Process**:
  1. Create child process using `fork()`
  2. In child: set up signal handlers, then execute command using `execvp()`
  3. In parent: wait for child completion and report exit status using `waitpid()`
- **Output Format:**

```
Executing: [command with arguments]
Child <pid> started
Child <pid> exited with status <exit_code>
```

Signal Handling Requirements

- **Signal Handler Function:**
  - Must call `exit(124)` when `SIGALRM` is received
  - Handler must NOT return - only exit
- **Registration (Required for Grading):**
  - Use `signal(SIGALRM, signal_handler)` in child process
  - This exact pattern must be present in your code
- **Timeout Setup:**
  - Call `alarm(3)` to set 3-second timeout
  - Place after signal registration, before `execvp()`
- **Process Flow:**
  - Child: Register handler → Set alarm → Execute command
  - Timeout result: Child terminated by SIGALRM [14]
  - Normal result: Child exits with command's status

Exit Status Interpretation

- **Status 0**: Successful command execution (echo, true, sleep when completed)
- **Status 1**: Command executed but reported failure (false command)
- **Status 127**: Command not found (execvp() failed)
- **Signal Termination**: Process killed by SIGALRM (no exit status, shows signal number)

Example Input/Output

1. Sample Input (`input1.txt` - Valid Commands):

```
echo hello
sleep 1
/bin/pwd
sleep 5
```

1. Expected Output:

```
Starting Process Controller
Executing: echo hello
Child <1234> started
Child <1234> exited with status 0
Executing: sleep 1
Child <1235> started
Child <1235> exited with status 0
Executing: /bin/pwd
Child <1236> started
Child <1236> exited with status 0
Executing: sleep 5
Child <20683> started
Child <20683> terminated by SIGALRM [14] - execution exceeded 3 seconds
All processes completed
```

2. Sample Input (`input2.txt` - Invalid Commands):

```
invalidcommand
nonexistentprog
```

2. Expected Output:

```
Starting Process Controller
Executing: invalidcommand
Child <1238> started
Child <1238> exited with status 127
Executing: nonexistentprog
Child <1239> started
Child <1239> exited with status 127
All processes completed
```

STEP 1 - Manual testing:

Ensure the following files are located in the **same directory**:

- **Your implementation**: `submission_HW3.c`
- The provided driver code: `driver.c`
- The provided `Makefile` (used by the autograder)
- The provided header file: `process_controller.h`
- The provided testcase folder: `Testing/`

```
# Build your application executable
make rebuild

# Run your implementation executable with a test input file.
# You may also run with additional test case files (input2.txt, input3.txt).
# Example (shown with input1.txt):
./run Testing/Testcases/input1.txt > student_output.txt

# ⚠️ Normalize the process IDs in student_output.txt.
# (The autograder replaces all actual PIDs with <PID> before comparison.)
# For example, your output:
#   "Child <7615> exited with status 0"
# will match the expected pattern:
#   "Child <PID> exited with status 0"

# Compare your output with the expected output:
diff student_output.txt Testing/Expected_Output/output1.txt
```

STEP 2 - Autograder testing:

Ensure the following files are located in the **same directory**, and then run the **Autograder**:

- **Your implementation file**: `submission_HW3.c`
- The provided driver code: `driver.c`
- The provided `Makefile` (used by the autograder)
- The provided header file: `process_controller.h`
- The provided testcase folder: `Testing/`
- The provided autograder script: `autograder_HW3.sh`

```
# Run the autograder
./autograder_HW3.sh
```

- **Check your grade** and fix any issues
- **Repeat until** you achieve the desired score

STEP 3 - Batch Autograder testing:

The batch autograder, **used by the instructor**, processes all student submissions at once. It utilizes the provided framework files, extracts the required files from your submitted `.zip` file, and performs grading accordingly.

⚠️ **Backup Files Before Running Batch Grader - The script deletes existing files to rebuild the environment from scratch.**

Ensure the following files are located in the **same directory**, and then run the **Batch Autograder**:

- **Your implementation file**: `submission_HW3.c`
- The provided driver code: `driver.c`
- The provided `Makefile` (used by the autograder)
- The provided header file: `process_controller.h`
- The provided testcase folder: `Testing/`
- The provided autograder script: `autograder_HW3.sh`
- The provided **Batch Autograder** script: `batchgrader_HW3.sh`
- **Your final submission ZIP file** consisting the required files for the submission: example, `Harry_Potter.zip`

```
# Run the batch-autograder
./batchgrader_HW3.sh
```

Final Testing requirements:

- ⚠️ **MUST test on ocelot server**: `ocelot-bbhatkal.aul.fiu.edu`
- ⚠️ **Test thoroughly before submission** - no excuses accepted
- ⚠️ **"Works on my computer" is NOT accepted** as an excuse
- ✅ **If you pass all test cases on the server, you will likely pass instructor's final grading**
- ✅ **What you see is what you get** - autograder results predict your final grade

---

☑ Section 6: Submission Requirements

File Requirements (⚠️ exact names required)

- **submission_HW3.c**: Your complete implementation
- **README.txt or README.pdf**: Student information and documentation
    - Your README ⚠️ **MUST include**:

```
# Student Information
- Full Name: [Your Full Name]
- PID: [Your FIU Panther ID]
- Section: [Your Course Section]
- FIU Email: [Your @fiu.edu email]

# Homework: Process Controller
[Brief description of your implementation approach]
[Mention which lecture examples you adapted and how]
```

Deliverables `ZIP` File Submission (⚠️ exact structure required - no additional files/subfolders)

> ⚠️ **DO NOT submit any other files other than these required files. The autograder may treat any additional items in the ZIP file as invalid, which will result in a grade of zero:** submission_HW3.c, README
>
> 📁 All other required framework files and the testcases will be supplied by the instructor to the autograder during final grading.

- ⚠️ You are required to submit your work as **a single ZIP archive file**
- ⚠️ **Filename Format**: your `Firstname_Lastname.zip` (**exact format required**)
- 📁 **Contents**:
  - `submission_HW3.c`
  - `README`

```
Harry_Potter.zip
├── submission_HW3.c   # Your implementation
└── README                 # Your details and brief implementation approach
```

☑️ Section 7: Grading Criteria

🤖 Autograder Testing

- ⚠️ Your implementation `submission_HW3.c` will be tested against the provided test cases as well as additional instructor test cases using the autograder.
- ⚠️ **Exact output matching required** - any deviation results in point deduction
- ⚠️ **Program must compile** without errors or warnings

Test Case Distribution:

- **Test 1 (30 points)**: Valid commands only - tests successful process execution
- **Test 2 (30 points)**: Invalid commands only - tests error handling
- **Test 3 (40 points)**: Mixed valid/invalid commands - tests comprehensive functionality

⛔ Penalties

- ⚠️ **Missing README**: **-10 points**
- ⚠️ **Missing** `submission_HW3.c`: **ZERO grade** (autograder compilation failure)
- ⚠️ **Incorrect ZIP filename**: **ZERO grade** (autograder compilation failure)
- ⚠️ **Wrong source filename**: **ZERO grade** (autograder compilation failure)
- 🔒 **Resubmission**: **NOT ALLOWED**

☑️ Section 8: Academic Integrity

- 👤 **This is an individual assignment**
- 💬 **You may discuss concepts** but not share code
- 📝 **All submitted code** must be your original work
- 📚 **You are encouraged to adapt from instructor provided coding examples** but must understand and modify them appropriately
- ⚠️ **Plagiarism** is a serious offense and will result in penalties - **ZERO grade** (academic integrity violation).

🚀 **Good luck with your homework!**