Project Report: Multi-Process System Monitor

1. Introduction

1.1. Background

In computing, monitoring multiple processes simultaneously can be crucial for effective system management. This project involves creating a multi-process system monitor that utilizes child processes to perform specific tasks and communicates their results back to a parent process.

1.2. Objectives

The main objectives of this project are:

- **To spawn multiple child processes**: The system will create several child processes to handle tasks in parallel.
- **To collect and manage data from these processes**: Each child process will produce data that needs to be gathered and processed by the parent.
- **To display the collected data**: The parent process will collect and display the results from all child processes, providing a unified view.

2. System Design

2.1. Architecture

The architecture of the system includes two main components:

- **Parent Process**: Responsible for creating and managing child processes. It sets up communication channels (pipes) and collects data produced by the child processes.

- **Child Processes**: Each child performs specific tasks. It writes its results to a pipe that the parent process can read from.

2.2. Technology Stack

- **Programming Language**: C, which is used for its low-level process and system management capabilities.
- **Libraries/Headers**:
  - sys/types.h: Defines types used in system calls.
  - unistd.h: Provides access to the POSIX operating system API.
  - sys/wait.h: Used for process control.
  - stdio.h and stdlib.h: For standard input/output and utility functions.
  - string.h: For string manipulation functions.

3. Implementation

3.1. Code Breakdown

- **Header File (system_monitor.h)**: Contains definitions for constants and function prototypes. This file sets up the groundwork for the system's functions and constants like the number of child processes and buffer sizes.

  >Header File (system_monitor.h):

  ```
  #ifndef system_monitor_h
  #define system_monitor_h
  ```

```
#include <sys/types.h>

// Define constants
#define NUM_CHILD_PROCESSES 3
#define BUFFER_SIZE 256

// Function prototypes
void start_monitoring();
void handle_child_process(int child_index, int pipe_fd);

#endif // SYSTEM_MONITOR_H
```

- **(system_monitor.c)**:
  - **start_monitoring() Function**:
    - Creates pipes for inter-process communication (IPC).
    - Forks child processes, each of which will execute a specific task.
    - Collects data from these child processes by reading from the pipes.
    - Ensures all child processes complete before the parent process exits.

>(system_monitor.c):

```
#include "system_monitor.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>

void start_monitoring() {
    int pipe_fd[NUM_CHILD_PROCESSES][2];
    pid_t child_pid;

    // Create pipes for IPC
    for (int i = 0; i < NUM_CHILD_PROCESSES; i++) {
        if (pipe(pipe_fd[i]) == -1) {
            perror("pipe");
            exit(EXIT_FAILURE);
        }
    }

    // Create child processes
    for (int i = 0; i < NUM_CHILD_PROCESSES; i++) {
        if ((child_pid = fork()) == -1) {
            perror("fork");
            exit(EXIT_FAILURE);
```

```
    } else if (child_pid == 0) {
        // Child process
        close(pipe_fd[i][0]); // Close read end of the pipe
        handle_child_process(i, pipe_fd[i][1]);
        exit(EXIT_SUCCESS);
    } else {
        // Parent process
        close(pipe_fd[i][1]); // Close write end of the pipe
    }
}

// Parent process: read from pipes
char buffer[BUFFER_SIZE];
for (int i = 0; i < NUM_CHILD_PROCESSES; i++) {
    ssize_t bytes_read;
    printf("Output from child process %d:\n", i);
    while ((bytes_read = read(pipe_fd[i][0], buffer, sizeof(buffer) - 1))
       > 0) {
        buffer[bytes_read] = '\0'; // Null-terminate string
        printf("%s", buffer);
    }
    close(pipe_fd[i][0]);
}
```

```
    // Wait for all child processes to complete
    for (int i = 0; i < NUM_CHILD_PROCESSES; i++) {
        wait(NULL);
    }
}
```

. **Main File:**

- **Entry Point**: The main function is the starting point of the program.
- **Header Inclusion**: #include "system_monitor.h" includes necessary declarations for start_monitoring().
- **Function Call**: start_monitoring() initiates the multi-process monitoring.
- **Exit Status**: return 0; indicates successful execution.
- **Program Flow**: Starts execution at main, calls start_monitoring(), and exits with a success code.

```
#include "system_monitor.h"

int main() {
    start_monitoring();
    return 0;
}
```

- **Child Handler File (handle_child_process.c)**:
    - **handle_child_process() Function**:
        - Redirects the output of the child process to the pipe.
        - Performs a task (this task could be any operation like running a system command).
        - Handles any errors that occur during the execution.

>Child Handler File (handle_child_process.c):

```
#include "system_monitor.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

void handle_child_process(int child_index, int pipe_fd) {
    // Redirect standard output to the pipe
    close(pipe_fd);
    dup2(pipe_fd, STDOUT_FILENO);
    close(pipe_fd);
```

```
    // Perform the task; replace with your specific task
    execlp("ls", "ls", NULL);

    // If execlp fails
    perror("execlp");
    exit(EXIT_FAILURE);
}
```

Tasks.json:

```
{
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: gcc-12 build active file",
            "command": "/usr/bin/gcc-12",
            "args": [
                "-fdiagnostics-color=always",
                "-g",
                "${workspaceFolder}/*.c",
                "-o",
                "${fileDirname}/main"
            ],
            "options": {
```

```
            "cwd": "${fileDirname}"
        },
        "problemMatcher": [
            "$gcc"
        ],
        "group": {
            "kind": "build",
            "isDefault": true
        },
        "detail": "Task generated by Debugger."
    }
  ],
  "version": "2.0.0"
}
```

3.2. Key Functions

- **start_monitoring()**:
  - Sets up pipes to facilitate communication between the parent and child processes.
  - Uses fork() to create multiple child processes. Each child process executes a designated task and writes the output to its respective pipe.

    - The parent process reads from the pipes and collects the output from each child process.
- **handle_child_process()**:
    - Redirects the standard output of the child process to a pipe, allowing the parent process to capture this output.
    - Executes the specific task assigned to the child process. This task could involve running a command or performing some computation.

4. Testing

4.1. Test Plan

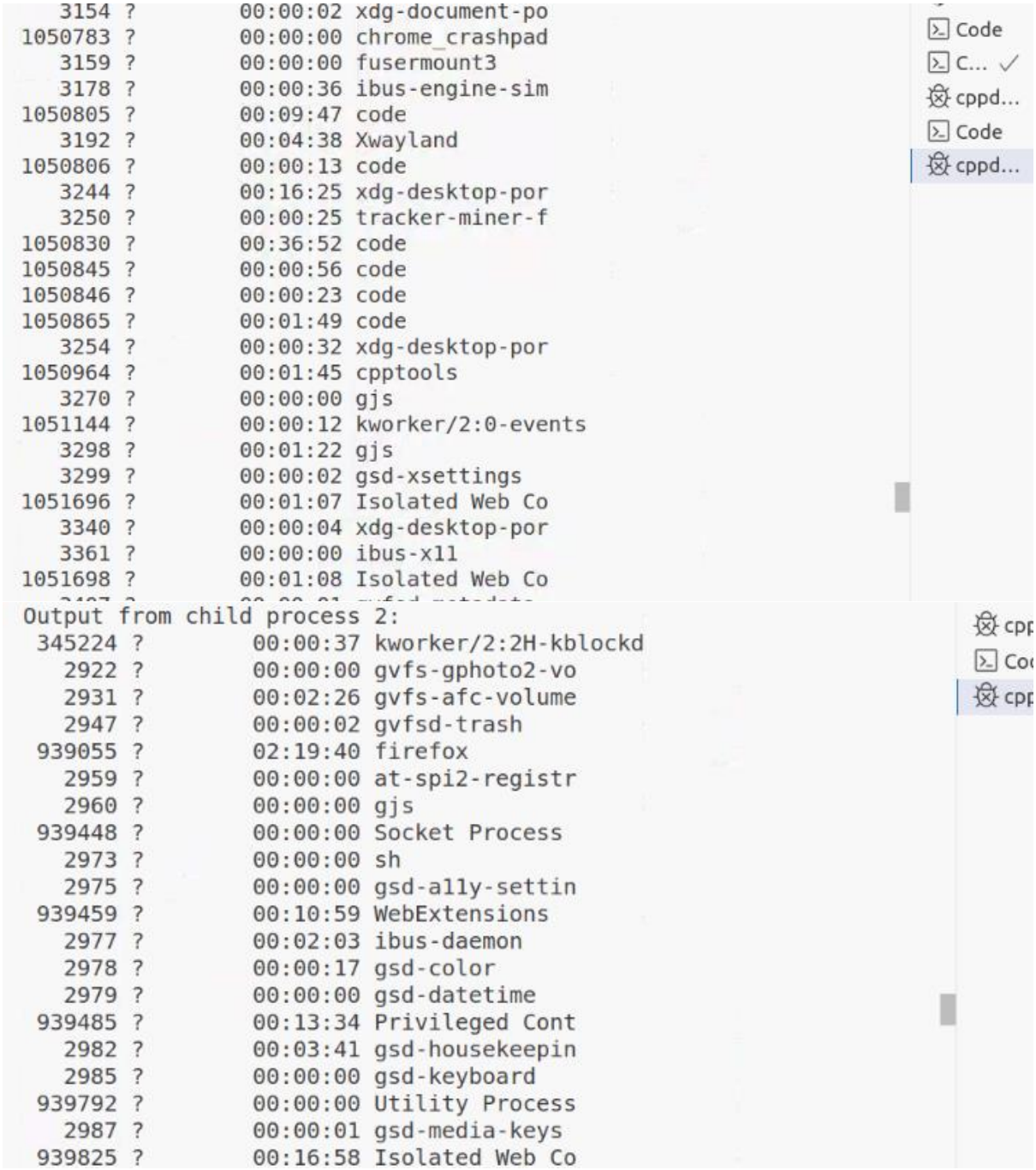
Testing involved:

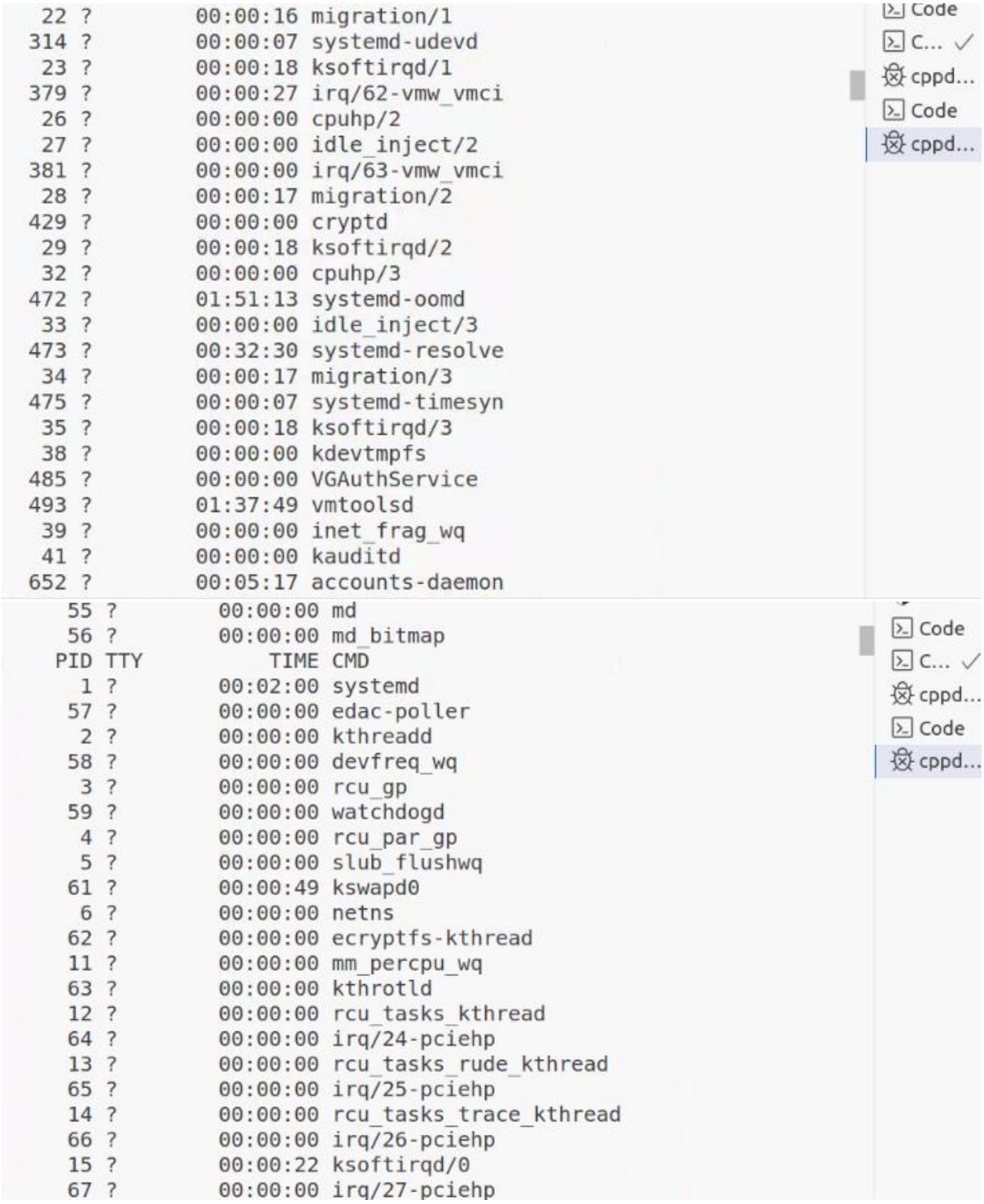
- Verifying that pipes are created successfully and that data is correctly passed between processes.
- Ensuring child processes are correctly spawned and complete their tasks.
- Checking that the parent process collects and displays data from all child processes accurately.

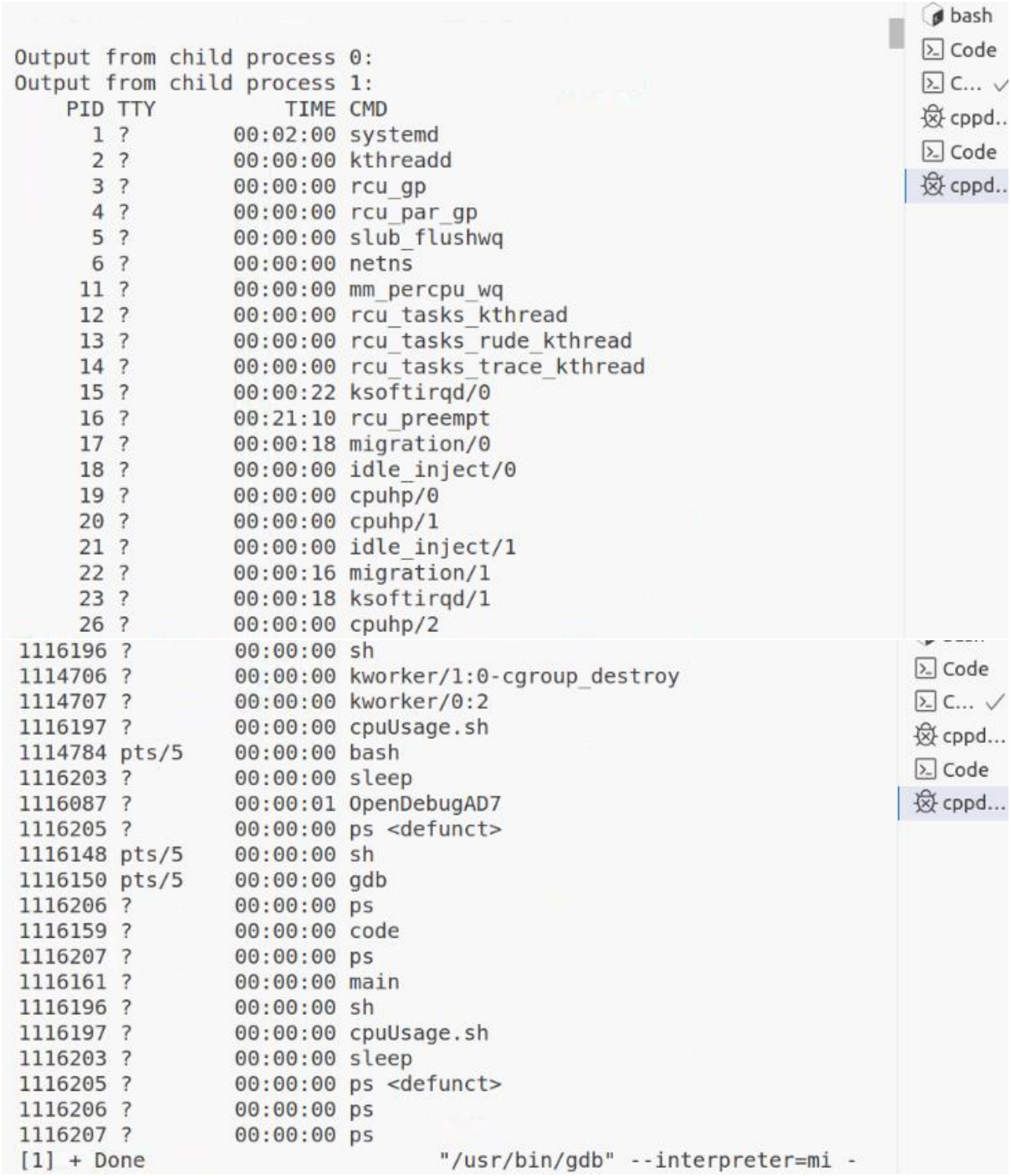
4.2. Results

- **Functionality**: The monitor successfully managed multiple child processes and collected data. It demonstrated effective use of pipes for IPC.
- **Observations**: The system efficiently handled concurrent tasks and provided a consolidated view of the data from multiple child processes.

Output:


3154 ? 00:00:02 xdg-document-po
1050783 ? 00:00:00 chrome_crashpad
3159 ? 00:00:00 fusermount3
3178 ? 00:00:36 ibus-engine-sim
1050805 ? 00:09:47 code
3192 ? 00:04:38 Xwayland
1050806 ? 00:00:13 code
3244 ? 00:16:25 xdg-desktop-por
3250 ? 00:00:25 tracker-miner-f
1050830 ? 00:36:52 code
1050845 ? 00:00:56 code
1050846 ? 00:00:23 code
1050865 ? 00:01:49 code
3254 ? 00:00:32 xdg-desktop-por
1050964 ? 00:01:45 cpptools
3270 ? 00:00:00 gjs
1051144 ? 00:00:12 kworker/2:0-events
3298 ? 00:01:22 gjs
3299 ? 00:00:02 gsd-xsettings
1051696 ? 00:01:07 Isolated Web Co
3340 ? 00:00:04 xdg-desktop-por
3361 ? 00:00:00 ibus-x11
1051698 ? 00:01:08 Isolated Web Co
Code
C...
cppd...
Code
cppd...
Output from child process 2:
345224 ? 00:00:37 kworker/2:2H-kblockd
2922 ? 00:00:00 gvfs-gphoto2-vo
2931 ? 00:02:26 gvfs-afc-volume
2947 ? 00:00:02 gvfsd-trash
939055 ? 02:19:40 firefox
2959 ? 00:00:00 at-spi2-registr
2960 ? 00:00:00 gjs
939448 ? 00:00:00 Socket Process
2973 ? 00:00:00 sh
2975 ? 00:00:00 gsd-a11y-settin
939459 ? 00:10:59 WebExtensions
2977 ? 00:02:03 ibus-daemon
2978 ? 00:00:17 gsd-color
2979 ? 00:00:00 gsd-datetime
939485 ? 00:13:34 Privileged Cont
2982 ? 00:03:41 gsd-housekeepin
2985 ? 00:00:00 gsd-keyboard
939792 ? 00:00:00 Utility Process
2987 ? 00:00:01 gsd-media-keys
939825 ? 00:16:58 Isolated Web Co


22 ? 00:00:16 migration/1
314 ? 00:00:07 systemd-udevd
23 ? 00:00:18 ksoftirqd/1
379 ? 00:00:27 irq/62-vmw_vmci
26 ? 00:00:00 cpuhp/2
27 ? 00:00:00 idle_inject/2
381 ? 00:00:00 irq/63-vmw_vmci
28 ? 00:00:17 migration/2
429 ? 00:00:00 cryptd
29 ? 00:00:18 ksoftirqd/2
32 ? 00:00:00 cpuhp/3
472 ? 01:51:13 systemd-oomd
33 ? 00:00:00 idle_inject/3
473 ? 00:32:30 systemd-resolve
34 ? 00:00:17 migration/3
475 ? 00:00:07 systemd-timesyn
35 ? 00:00:18 ksoftirqd/3
38 ? 00:00:00 kdevtmpfs
485 ? 00:00:00 VGAuthService
493 ? 01:37:49 vmtoolsd
39 ? 00:00:00 inet_frag_wq
41 ? 00:00:00 kauditd
652 ? 00:05:17 accounts-daemon
55 ? 00:00:00 md
56 ? 00:00:00 md_bitmap
PID TTY TIME CMD
1 ? 00:02:00 systemd
57 ? 00:00:00 edac-poller
2 ? 00:00:00 kthreadd
58 ? 00:00:00 devfreq_wq
3 ? 00:00:00 rcu_gp
59 ? 00:00:00 watchdogd
4 ? 00:00:00 rcu_par_gp
5 ? 00:00:00 slub_flushwq
61 ? 00:00:49 kswapd0
6 ? 00:00:00 netns
62 ? 00:00:00 ecryptfs-kthread
11 ? 00:00:00 mm_percpu_wq
63 ? 00:00:00 kthrotld
12 ? 00:00:00 rcu_tasks_kthread
64 ? 00:00:00 irq/24-pciehp
13 ? 00:00:00 rcu_tasks_rude_kthread
65 ? 00:00:00 irq/25-pciehp
14 ? 00:00:00 rcu_tasks_trace_kthread
66 ? 00:00:00 irq/26-pciehp
15 ? 00:00:22 ksoftirqd/0
67 ? 00:00:00 irq/27-pciehp
Code
C...
cppd...
Code
cppd...
Code
C...
cppd...
Code
cppd...




```
Output from child process 0:
Output from child process 1:
    PID TTY          TIME CMD
      1 ?        00:02:00 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
      4 ?        00:00:00 rcu_par_gp
      5 ?        00:00:00 slub_flushwq
      6 ?        00:00:00 netns
     11 ?        00:00:00 mm_percpu_wq
     12 ?        00:00:00 rcu_tasks_kthread
     13 ?        00:00:00 rcu_tasks_rude_kthread
     14 ?        00:00:00 rcu_tasks_trace_kthread
     15 ?        00:00:22 ksoftirqd/0
     16 ?        00:21:10 rcu_preempt
     17 ?        00:00:18 migration/0
     18 ?        00:00:00 idle_inject/0
     19 ?        00:00:00 cpuhp/0
     20 ?        00:00:00 cpuhp/1
     21 ?        00:00:00 idle_inject/1
     22 ?        00:00:16 migration/1
     23 ?        00:00:18 ksoftirqd/1
     26 ?        00:00:00 cpuhp/2
1116196 ?        00:00:00 sh
1114706 ?        00:00:00 kworker/1:0-cgroup_destroy
1114707 ?        00:00:00 kworker/0:2
1116197 ?        00:00:00 cpuUsage.sh
1114784 pts/5    00:00:00 bash
1116203 ?        00:00:00 sleep
1116087 ?        00:00:01 OpenDebugAD7
1116205 ?        00:00:00 ps <defunct>
1116148 pts/5    00:00:00 sh
1116150 pts/5    00:00:00 gdb
1116206 ?        00:00:00 ps
1116159 ?        00:00:00 code
1116207 ?        00:00:00 ps
1116161 ?        00:00:00 main
1116196 ?        00:00:00 sh
1116197 ?        00:00:00 cpuUsage.sh
1116203 ?        00:00:00 sleep
1116205 ?        00:00:00 ps <defunct>
1116206 ?        00:00:00 ps
1116207 ?        00:00:00 ps
[1] + Done                       "/usr/bin/gdb" --interpreter=mi -
```

5. Conclusion

5.1. Summary

The multi-process system monitor was successfully implemented, achieving its goals of managing and collecting data from child processes. It demonstrated the capability to handle multiple tasks simultaneously and provide a coherent view of the collected data.

5.2. Recommendations

- **Expand Functionality**: Future improvements could include monitoring additional types of metrics or tasks.
- **Improve Error Handling**: Enhance the system's robustness and provide more detailed error messages to improve user experience.