

Project 2

Mukul Mehar

Introduction

In this project report, I will discuss the methodologies used in this project to save and restore the program state in memory, specifically, the dynamically allocated anonymous memory, using a new system call, called *mmcontext*, instead of using the fork system call to do the same.

In the fork-based approach, we create another process and delegate unsafe code to the child. Since modifications performed by a child process are not visible to the parent, we can restore a safe program state by resuming in the parent after the child has finished executing unsafe code. However, this approach is not very efficient due to the overhead of creating an additional process and switching between parent and child.

To address this issue, a new system call is proposed to save the context of a process, specifically, the dynamically allocated anonymous memory.

The system call accepts only one argument called state, which can either be 0 or 1. If state is 0, then the system call saves the dynamically allocated anonymous memory to a new field in the task_struct called context. If state is 1, then the system call copies the memory back from the context field to the original position in the process address space.

Implementation Details

First of all, in order to store the state of a process, I have defined a new structure, *p_context*, whose *list_head* member is linked to the task_struct's *context field* (a new member of the task struct which I have defined).

The structure looks like this:

```
struct p_context {
    unsigned long addr;
    unsigned long size;
    void *buffer;
    struct list_head list;
```

```
};
```

Here, the *buffer* field of the struct holds the actual copied memory from the userspace. The *addr* field stores the corresponding userspace address (starting address) of the memory copied in the buffer, and the *size* field stores the size of buffer stored in *buffer*. The *list* field, as we saw earlier, is linked to the task struct's *context* field.

For storing the state of the process, I first copied the heap memory (*mm->brk - mm->start_brk*) into the *context* list, and then traversed the *vm_area_struct* of the process (using *mm → mmap*), to copy any anonymous memory that is not backed by file and is within the data segment of the process.

For copying memory from userspace to kernel, I have used *copy_from_user()* function.

For restoring the context, we just need to traverse the *context* list in the *task_struct* and copy the memory of *buffer* to the userspace address, pointed by the *addr* field of the *struct p_context*. We also need to deallocate the memory for the list node while restoring the context.

For copying memory kernel to userspace, I have used *copy_to_user()* function.

I have also added code to clean up the process state upon termination, if any process exits before restoring the context.

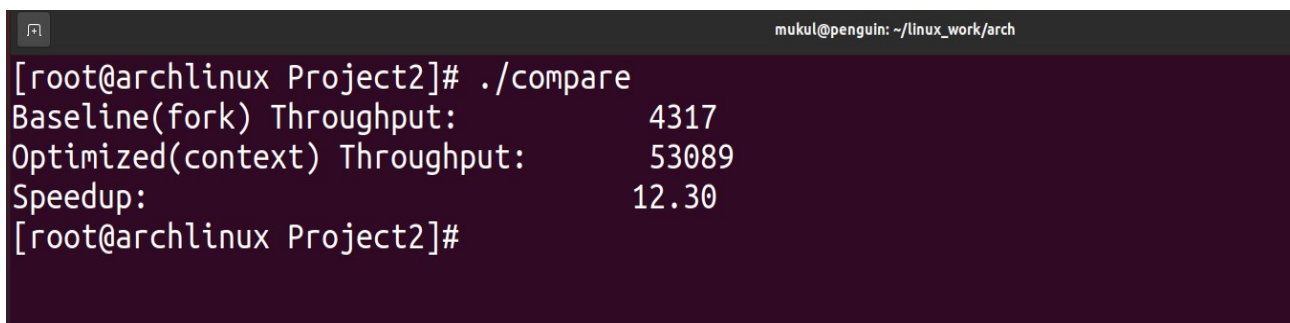
Benefits

Using this system call to save the context of a process, specifically, the dynamically allocated anonymous memory, has several benefits over using *fork*. First, it is more efficient, as it only copies the dynamically allocated memory, and not the entire memory of the process. This reduces the time taken for context switching and saves memory. Second, it allows for better

resource allocation, as the operating system can choose to restore only the required memory, rather than restoring the entire memory of the process. Third, it enables the operating system to implement features such as process checkpointing, where the state of a process can be saved and restored at a later time.

Results

Below are results of running the provided scripts.

A terminal window with a dark purple background. The title bar shows a window icon and the text 'mukul@penguin: ~/linux_work/arch'. The terminal content shows a shell prompt '[root@archlinux Project2]#', followed by the command './compare'. The output displays three lines: 'Baseline(fork) Throughput: 4317', 'Optimized(context) Throughput: 53089', and 'Speedup: 12.30'. The prompt '[root@archlinux Project2]#' appears again at the bottom.

```
mukul@penguin: ~/linux_work/arch
[root@archlinux Project2]# ./compare
Baseline(fork) Throughput:      4317
Optimized(context) Throughput: 53089
Speedup:                       12.30
[root@archlinux Project2]#
```

As we can see, that the new, *mmcontext*, system outperforms the fork based approach by quite a margin.

This analysis is only based on the time. But this new system call abstraction also saves on space, compared with the fork-based approach.

Conclusion

In this project report, we proposed a new system call to save the context of a process, specifically, the dynamically allocated anonymous memory. We discussed the benefits of using this system call over using fork, including efficiency, better resource allocation, and the ability to implement features such as process checkpointing.