CS 130: Project 2 - User Programs Design Document

Group Information

Please provide the details for your group below, including your group's name and the names and email addresses of all members.

- **Group Name**: *[printf("team")]*
- **Member 1**: Zirui Wang `<wangzr2024@shanghaitech.edu.cn>`
- **Member 2**: Xing Wu `<wuxing2024@shanghaitech.edu.cn>`

Preliminaries

> If you have any preliminary comments on your submission, notes for the TAs, or extra credit, please give them here.

Your answer here.

None.

Argument Passing

Data Structures

> **A1:** Copy here the declaration of each new or changed `struct` or struct member, global or static variable, `typedef`, or enumeration. Identify the purpose of each in 25 words or less.

None.

Algorithms

> **A2:** Briefly describe how you implemented argument parsing. How do you arrange for the elements of `argv[]` to be in the right order? How do you avoid overflowing the stack page?

Argument parsing happens in function `start_process()`, where we receive a pointer `file_name_`. We use `strtok_r()` to seperate the arguments, and push them into the stack with the help of `esp` pointer after a successful `load()`.

The arguments we separate are stored in the `char *argv[64];`. And we loop through it in reverse order. Thus they are pushed into the stack in right order.

We use two assertions to set a limit on the total length and number of arguments to avoid overflow.

Rationale

> **A3:** Why does Pintos implement `strtok_r()` but not `strtok()`?

`strtok_r()` is reentrant, which means we can call it by different threads simultaneously.

A4: In Pintos, the kernel separates commands into an executable name and arguments, while Unix-like systems have the shell perform this separation. Identify at least two advantages of the Unix approach.

- Easier Debugging and Testing: Since the shell is a separate program, it can be tested and debugged independently of the kernel. This makes it less risky and easier to fix, since kernel bugs might crash the entire system.
- Simplified Kernel Design: By offloading command parsing to the shell, the Unix kernel remains simpler and more efficient.
- Customizability and Extensibility: Since the shell is a user-space program, it can be easily replaced or customized. For example, a shell can have its own syntax rules.

System Calls

Data Structures

B1: Copy here the declaration of each new or changed `struct` or struct member, global or static variable, `typedef`, or enumeration. Identify the purpose of each in 25 words or less.

New Enumeration to `threads/thread.h`

```
#ifdef USERPROG
/* States when a process loads the executable. */
enum load_status
{
  LOAD_READY,   /* Not loading but ready to load. */
  LOAD_SUCCESS, /* Loading the executable. */
  LOAD_FAIL     /* Loading fails. */
};
#endif
```

New Struct Member to `struct thread`

```
struct thread
{
  /* ... */

#ifdef USERPROG
  /* Owned by userprog/process.c. */
  uint32_t *pagedir; /* Page directory. */

  struct thread *creator;          /* The thread that creates this thread. */
  enum load_status ch_load_status; /* Load status. */
  struct semaphore ch_load_sema;   /* Semaphore for loading. */

  struct list ch_exit_data; /* List of child threads. */

  struct file *exec_file; /* Loaded executable file. */
#endif

  /* ... */
}
```

New Struct to `userprog/process.h`

```
/* Used to track the exit value after a thread exits. */
struct exit_data
{
  tid_t tid;                 /* The thread id of the thread. */
  int exit_code;             /* The exit code of the thread. */
  bool called_process_wait;  /* Whether process_wait() has been called. */
  struct thread *father;     /* The father thread of the thread. */
  struct hash_elem hashelem; /* The hash element stored in hash_exit_data. */
  struct list_elem listelem; /* The list element stored in ch_exit_data. */
  struct semaphore die_sema; /* Wait for the thread to die. */
};
```

New Static Variable to `userprog/process.c`

```
/* A hash table to find exit data from tid. */
static struct hash hash_exit_data;
```

New Static Variable to `userprog/syscall.c`

```
/* file descriptor table. */
static struct bitmap *fd_table; /* Tracking allocated (1) and free (0) fds. */
static struct file *fd_entry[OPEN_FILE_MAX]; /* Maps fd to struct file *. */
static tid_t fd_owner[OPEN_FILE_MAX];        /* thread tid owning each fd. */
static struct lock fd_table_lock; /* Mutex protection for fd table access. */
```

New Global Variable to `filesys/filesys.c`

```
/* Prevent multiple threads to use filesystem at same time. */
struct lock filesys_lock;
```

B2: Describe how file descriptors are associated with open files. Are file descriptors unique within the entire OS or just within a single process?

In our implementation, file descriptors are managed through a global static structure consisting of a bitmap `fd_table` tracking allocated fds, an array `fd_entry` mapping fds to `struct file` pointers representing open files, and an ownership array `fd_owner` recording the thread TID associated with each fd, all protected by a global mutex lock `fd_table_lock`.

File descriptors are uniquely within entire OS, as the `fd_table` , `fd_entry` and `fd_owner` are shared across all processes, with the fd value acting as a global index into these structures.

Algorithms

B3: Describe your code for reading and writing user data from the kernel.

First of all, before dereferencing any pointer, we check that they points below `PHYS_BASE`. Any invalid pointer will terminate the process by invoking `exit_(-1)`.

Then we define a `READ` macro, calling `read_data()` to incrementally fetch arguments from the user stack.

If dereferencing user pointers cause a "page fault", it will be handled in `page_fault()`.

> **B4:** Suppose a system call causes a full page (4,096 bytes) of data to be copied from user space into the kernel. What is the least and the greatest possible number of inspections of the page table (e.g., calls to `pagedir_get_page()`) that might result? What about for a system call that only copies 2 bytes of data? Is there room for improvement in these numbers, and how much?

4096 bytes: Least: 1 inspection. (entire buffer in one valid page) Greatest: 4096 inspections. (each byte in a separate page)

2 bytes: Least: 1 inspection. (both bytes in the same page) Greatest: 2 inspections. (bytes in different pages)

Improvement: In pintos, a system call will only cause a continuous segment of data to be copied. Even if we have to copy 4096 bytes of data, the worst case is that they cross two pages. Therefore we only need 2 inspection in that case.

> **B5:** Briefly describe your implementation of the "wait"system call and how it interacts with process termination.

The "wait"system call invokes `process_wait()`. In pintos there is only one thread in a process, we can simply convert `pid_t` to `tid_t`.

We defined a new struct `exit_data`, to track the exit code of a thread. When a thread terminates, it assigns its exit code to its exit data and increments the semaphore. Then the parent process know it terminates and finish waiting.

> **B6:** Accessing user program memory at a user-specified address may fail due to a bad pointer value, requiring termination of the process. Describe your strategy for managing error-handling without obscuring core functionality and ensuring that all allocated resources (locks, buffers, etc.) are freed. Give an example.

For example, when we write data to files, a lock called `filesys_lock` is acquired by the crrent thread, it is released after executing write operation.

```
lock_acquire (&filesys_lock);
int ret = file_write (op_file->file, buffer, size);
lock_release (&filesys_lock);
```

However, when `file_write()` causes page fault, the `lock_release()` below it will not be performed. Therefore in page fault handler we have to release it if the dying process holds that lock.

Synchronization

> **B7:** The "exec"system call returns -1 if loading the new executable fails, so it cannot return before the new executable has completed loading. How does your code ensure this? How is the load success/failure status passed back to the thread that calls "exec"?

We use a semaphore to force the parent thread to wait until the new executable finishes loading.

There is a enumerate called `load_status` in each thread to track the load status. When the child finishes loading, it will modify its parent's load status.

> **B8:** Consider a parent process P with child process C. How do you ensure proper synchronization and avoid race conditions when:
>
> - P calls `wait(C)` before C exits?
> - P calls `wait(C)` after C exits?
> - P terminates without waiting, before C exits?
> - P terminates after C exits?
> - Are there any special cases?

- When P calls `wait(C)` before C exits: P will be forced to wait until C exits, then get the exit code.
- P calls `wait(C)` after C exits: P will immediately get the exit code and finish waiting.
- P terminates before C exits: The exit code doesn't matter in this case. P will free C's exit data and C won't access them when it exits.
- P terminates after C exits: The exit code doesn't matter in this case. P will free C's exit data and C won't access them when it exits.

Rationale

> **B9:** Why did you choose to implement access to user memory from the kernel in the way that you did?

We use the second approach since it is normally faster than the first one and tends to be used in real kernels. (according to the pintos document)

> **B10:** What advantages or disadvantages can you see to your design for file descriptors?

Advantages:

- Avoid per-process file descriptor table management overhead.
- TID-based ownership enforces access isolation between processes, preventing unauthorized operations.

Disadvantages:

- The global mutex Lock may lead to performance issues under heavy concurrency.

> **B11:** The default `tid_t` to `pid_t` mapping is the identity mapping. If you changed it, what advantages does your approach offer?

We don't change it because every process in pintos only has one thread. The identity mapping is reasonable and efficient.

Survey Questions

> Do you have any suggestions for the TAs to more effectively assist students, either for future quarters or the remaining projects? Any other comments?

None.