Pintos Project 2 Report: User programs

Group members: 11612132 李润林 11610101 韦青茂

Task 1: Argument Passing

I. Data structures and functions

In thread.h:

```
int exit_error;
```
判断是否已经执行的 flag
```
bool load_success;
```
判断 ELF 是否执行的 flag
```
struct thread* parent;
```
一个用来储存父进程的 thread 对象
```
struct semaphore waiting_child;
```
一个判断该进程是否在等待子进程完成的 semaphore
```
int wait_on;
```
储存该进程正在等待的进程

In thread.c:

```
struct child_process* c
```
一个 child_process 对象的指针
```
list_push_back (&running_thread()->childs, &c->elem);
```
将上述指针以及指向对象的头指针传入 list 储存

In thread.c:

In process_execute():
修改该方法，分割 filename 并且储存的 strtok_r，以及加入等待子进程唤醒和判断子进程是否成功唤醒的方法调用。
Update：在该方法中加入了复制并分割的操作，以规避 exec-bound-2 直接调用出现内核访问错误的问题

In start_process():
修改方法，复制了命令并且进行分割储存，同时计算参数个数。
加入对唤醒不成功的处理，将重新唤醒父进程
通过说明文档给出的规则，使用 esp 进行压栈。

In process_wait():
查找子进程，并且增加子进程只可唤醒一次条件判断，在唤醒结束后杀死进程。

In syscall.c:

修改并完成 handler,其中的 write 功能来自 task3，与 task1 有关联因此在此提及

II. Algorithms and implementations

1.要求

Task1 要求我们获取命令行参数并且压栈储存，我们首先需要将获取到的大字符串进行分割，把单个参数分割出来知乎使用 esp 指针进行压栈操作。

File_name 的传入大概为 name.c argv argv....

2. 实现

首先我们复制并且分割传入的 filename 字符串，在获取了文件名以及各个参数的情况下计算参数数量，之后从 esp 指针下读取数据，分配给 argv 指针以变量数 argc 个地址空间，将每一个分割的参数地址和 argv 压入栈中即算完成

III. Synchronization

该操作最大的问题处在进程间的关系问题上，我们不希望线程在唤醒，创建环节出现未完成的问题，因此我们需要父进程实现等待，在引入 semaphore 规避上述问题的基础上，之后我们也需要对 semaphore 本身降低以防止父进程阻塞。Semaphore 只有子进程完成 argv 传递后才会被恢复保证父进程的正常运行。

Task 2: Process Control Syscalls

I. Data structures and functions

In syscall.c:

```
struct file* process_get_file
```

从当前进程打开的文件中找到对应的指针

```
void process_close_file( int fd)
```

关闭当前进程打开的文件

```
void* check_addr(const void *vaddr)
```

检查地址是否为用户空间地址，检查 page 是否存在，并且转换为内核地址传入

```
void get_arg (struct intr_frame *f, int *arg, int n)
```

从 esp 指针获取参数，检查参数指针合法性

syscall_handler()
We use switch in this method to implement halt,exit,exec,wait,create,remove,open,filesize,read,write,seek,close,tell.

```
void check_valid_string (const void* str)
```
检查字符串合法性，包括空字符串检查以及字符串地址合法性检查

```
struct child_process* get_child_process (int tid)
```
先行找出当前子进程对应 tid 进程

```
void exit_proc(int status)
```
退出函数，在退出时如果父进程等待则唤醒父进程

In process.c:

```
void process_wait (tid_t child_tid)
```
此方法实现进程等待

```
void process_exit (void)
```
此方法实现进程结束

II. Algorithms and implementations

task 2 主要需要处理并获取上个 task 中压入栈的信息，并基于此完成相应的系统调用，包括了关闭系统，结束进程，等待进程以及执行进程等。
在 handler 中完成了 halt 以及 exit，exec，wait
Halt 直接根据文档调用 shutdown_power_off()
Exit 使用实现的 get_arg 获取当前的退出返回状态,之后调用 exit_proc()来进行退出,在 exit_proc（）中，我们首先退出子进程，之后判断当前父进程是否在等待，等待则唤醒父进程，之后修改 semaphore 以便父进程正常运行，输出当前结束状态以符合要求。之后调用 thread_exit（），在进行文件列表，父子列表回收之后结束。
Exec 在复制并且分割读取到文件名之后打开为 file 类对象，如果文件名查找为-1 则直接解锁并返回-1，否则执行 open 并且在结束时关闭流并解锁
Wait 调用了 process_wait（），在 process_wait 中首先根据 tid 查找子进程，找不到直接返回-1，找到之后判断该子进程是否被唤醒过，如果未被唤醒过啧则唤醒并且提升 semaphore，之后从表中移除该子进程并且返回退出值。

III. Synchronization

该 task 要求多层调用的次数较多，同时对每次调用时需要执行的操作的逻辑顺序要求较大，尽可能的将方法放在一个 method 下有利于修改以及查错
同时由于文件系统需要保证访问的独立，需要引入锁以保证进程间的文件系统访问互斥

Task 3: File Operation Syscalls

I.Data structures and functions

In syscall.c:

```
struct proc_file {
    struct file* ptr;
    int fd;
    struct list_elem elem;
};
```

将文件内容储存，包括了数量指针，文件指针以及打开文件列表

```
static void syscall_handler (struct intr_frame *f);
```

同样是在 handler 中实现，包括了：

```
case SYS_CREATE:
case SYS_REMOVE:
case SYS_OPEN:
case SYS_FILESIZE:
case SYS_READ:
case SYS_WRITE:
case SYS_SEEK:
case SYS_CLOSE:
case SYS_TELL:
```

实现了 task 3 的要求，以及部分辅助完成的函数，部分对前文 task 有帮助。

以及部分 handler 外定义的的在 handle 时使用的函数：

```
struct file* process_get_file (int fd)
void process_close_file( int fd)
void* check_addr(const void *vaddr)
void get_arg (struct intr_frame *f, int *arg, int n)
void check_valid_string (const void* str)
int exec_proc(char *file_name)
void exit_proc(int status)
```

In thread.h:

```
struct list files;
```

记录该线程打开的文件

II. Algorithms and implementations

在完全理解 task1,2 的基础上，我们可以进行 syscall 的处理。我们的输入存放在了栈中，当调用时进行访问，首先就需要判断命令，地址，参数的合法性，之后访问同一个 handler，在 handler 中决

定具体调用的 function。调用时上锁并在完成或错误时解锁。
Create 首先获取文件名并判断字符串合法性，不合法 return-1 在合法的情况下上锁并且执行 filesys_create 创建文件，在完成后解锁。
Remove 首先获取文件名并判断字符串合法性，合法则上锁并且使用 filesys_remove，成功会返回 true，否则返回 false
Open 首先获取文件名并判断字符串合法性，合法则获得锁，尝试调用 filesys_open，打开失败返回 -1 并解锁并终止，成功则返回文件信息，并且将改文件推入 list。之后释放锁。
Filesize 首先获取文件名并判断字符串合法性，合法则获得锁。尝试调用 process_get_file，失败则返回-1 并解锁，成功则调取 file_length 并解锁，返回获得长度。
Read 首先获取文件名并判断字符串合法性，合法则获得锁。之后根据 fd 调取 process_get_file，失败返回-1 并解锁，成功调取 file_read 并解锁
Write 首先获取文件名并判断字符串合法性，合法则获得锁。之后根据 fd 调取 process_get_file，失败返回-1 并解锁，成功调取 file_write 并解锁
Seek 首先获取文件名并判断字符串合法性，合法则获得锁。之后根据 fd 调取 process_get_file，失败返回-1 并解锁，成功调取 file_seek 并解锁
Close 首先获取文件名并判断字符串合法性，合法则获得锁。之后在当前打开文件列表中存在指定元素时，关闭该元素并从表中移除并解锁。
Tell 首先获取文件名并判断字符串合法性，合法则获得锁。之后根据 fd 调取 process_get_file，失败返回-1 并解锁，成功调取 file_tell 并解锁

III. Synchronization

Task3 考验的更多的是对方法参数的处理以及传递，我们需要注意合法性以及变量的同步问题。

Questions:

A reflection on the project – what exactly did each member do? What went well, and what could be improved?
11612132 李润林：task 1 and report
11610101 韦青茂：task1 ~3

Does your code exhibit any major memory safety problems (especially regarding C strings), memory leaks, poor error handling, or race conditions?
我们处理了内核地址的转换问题，对地址、参数等合法性的判断

Did you use consistent code style? Your code should blend in with the existing Pintos code. Check your use of indentation, your spacing, and your naming conventions.
我们使用了正常的代码风格、正确的缩进并保持着良好的命名规范

Is your code simple and easy to understand?
我们的代码有着清晰的注释以及明确的命名，方法的逻辑清晰，方法间的调用明确

If you have very complex sections of code in your solution, did you add enough comments to explain them?
我们有着清晰的代码注释

Did you leave commented-out code in your final submission?
代码中的简单纠错被我们删除了，但是一些关键信息的纠错虽然被注释掉但是没有被删除，以便未来调整

Did you copy-paste code instead of creating reusable functions?
我们几乎没有重复代码，而是不断地进行重复调用，使得代码很简洁

Are your lines of source code excessively long? (more than 100 characters)
没有，我们的方法参数十分明确，逻辑按部就班，不会出现冗长的判断以及条件语句

Did you re-implement linked list algorithms instead of using the provided list manipulation
我们使用了提供的 link-list

reference：DNXie Liziwl https://github.com/liziwl/operating_system_project2