CS 302
Operating System

Project 2: User Program

Register team Due:	April 14, 2019
Report Due:	May 14, 2019
Code Due:	May 14, 2019

Contents

- **Project 2 Task Introduction**
  - Task 1: Argument Passing
  - Task 2: Process Control Syscalls
  - Task 3: File Operation Syscalls
- **Report**
  - What we focus in report
- **Register team**

Task 1: Argument Passing

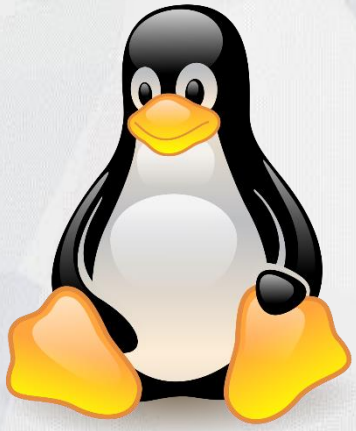
```
/* Starts a new thread running a user program
loaded from FILENAME.
The new thread may be scheduled (and may even exit)
before process_execute() returns.
Returns the new process's thread id, or TID_ERROR
if the thread cannot be created. */

tid_t
process_execute (const char *file_name){
  ……
}
```

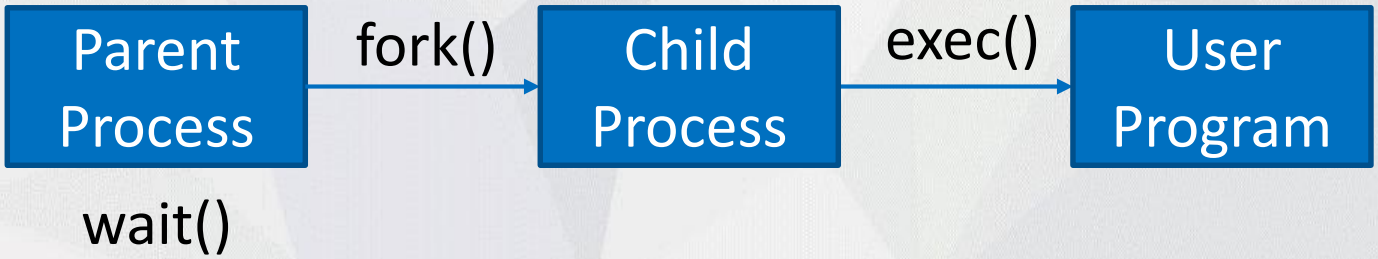
Call process_execute (“ls -al”) will provide the 2 arguments,[“ls”, “-al”] to the user program.

Execute user program

fork() + exec*() + wait()



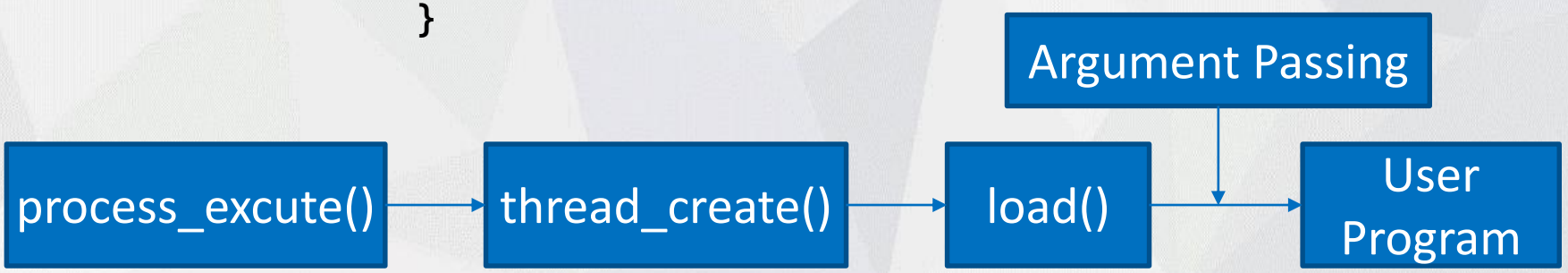
```
 1  int system_ver_CS302(const char *cmd_str) {
 2      if(cmd_str == -1)
 3          return -1;
 4      if(fork() == 0) {
 5          execl(cmd_str, cmd_str, NULL);
 6          fprintf(stderr,
                "%s: command not found\n", cmd_str);
 7          exit(-1);
 8      }
 9      wait(NULL);
10      return 0;
11  }
```



Execute user program

Pintos

```
tid_t process_execute (const char *file_name) {
    ……
  /* Create a new thread to execute FILE_NAME. */
  tid = thread_create (file_name_, PRI_DEFAULT,
        start_process, fn_copy);
    ……
}
/* A thread function that loads a user process and
starts it running. */
static void start_process (void *file_name_)
{
    ……
    /* Loads an ELF executable from FILE_NAME into
       the current thread. */
    success = load (file_name, &if_.eip, &if_.esp);
    ……
}
```

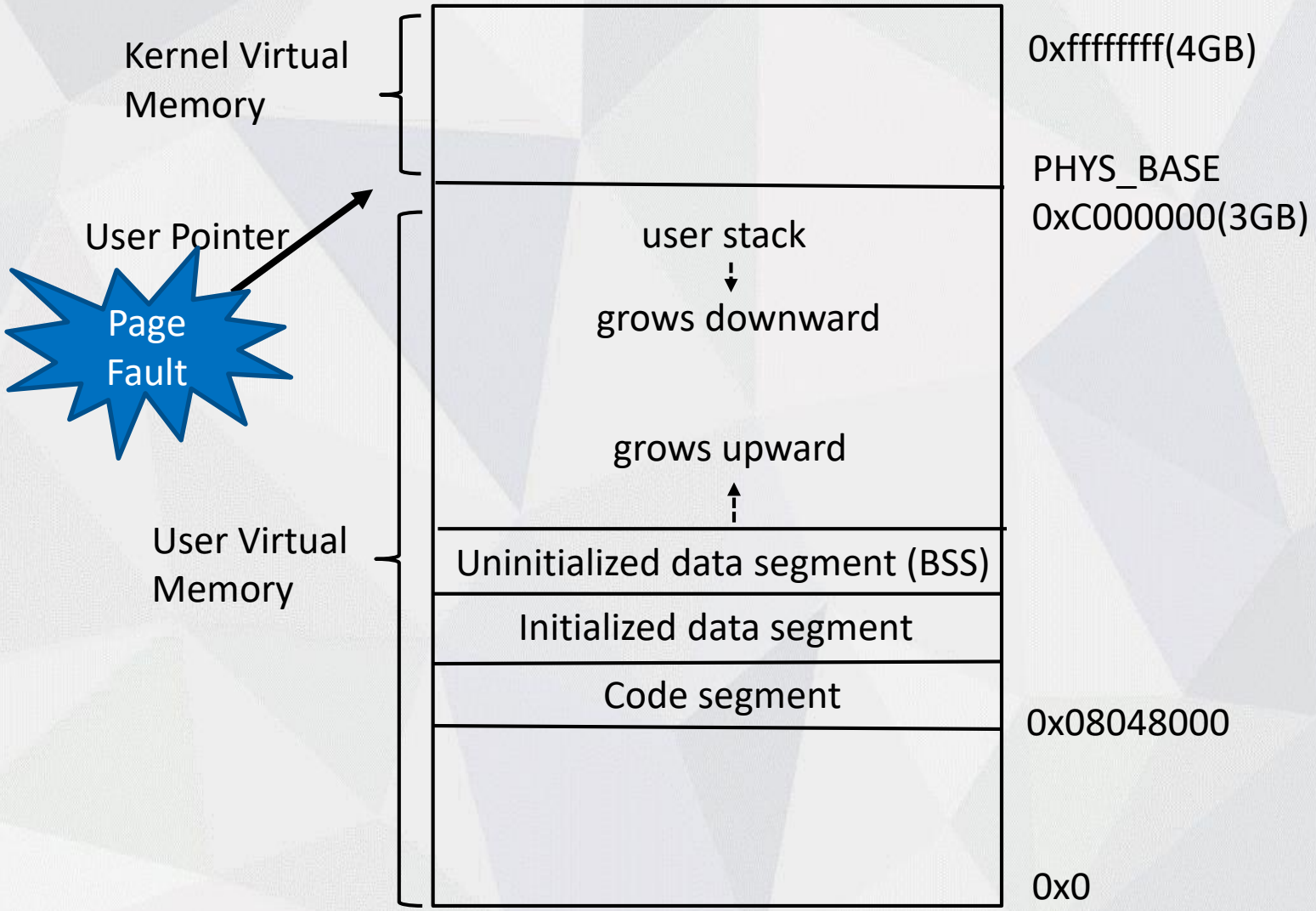


Argument Passing in C Language

```c
int main (int argc, char *argv[])
{
    printf("argc is %d\n", argc);
    int i;
    for (int i=0; i<argc; ++i) {
       printf("argv[%d] %s\n", i, argv[i]);
    }
    return 0;
}
```

```
$ ./a.out a b c
argc is 4
argv[0] a.out
argv[1] a
argv[2] b
argv[3] c
$
```

Virtual Memory Layout



Program Startup

For example command: “/bin/ls –l foo bar”

Address	Name	Data	Type
0xbffffffc	argv[3][...]	bar\0	char[4]
0xbffffff8	argv[2][...]	foo\0	char[4]
0xbffffff5	argv[1][...]	-l\0	char[3]
0xbfffffed	argv[0][...]	/bin/ls\0	char[8]
0xbfffffec	word-align	0	uint8_t
0xbfffffe8	argv[4]	0	char *
0xbfffffe4	argv[3]	0xbffffffc	char *
0xbfffffe0	argv[2]	0xbffffff8	char *
0xbfffffdc	argv[1]	0xbffffff5	char *
0xbfffffd8	argv[0]	0xbfffffed	char *
0xbfffffd4	argv	0xbfffffd8	char **
0xbfffffd0	argc	4	int
0xbfffffcc	return address	0	void (*) ()

Arguments are pushed in **right-to-left** order.

Task 2: Process Control Syscalls

Pintos currently only supports one syscall: **exit**. You will add support for the following new syscalls: **halt**, **exec**, **wait**, and **practice**.

lib/user/syscall.h

```
void halt (void) NO_RETURN;
void exit (int status) NO_RETURN;
pid_t exec (const char *file);
```

practice： The practice syscall just adds 1 to its first argument, and returns the result

Task 3: File Operation Syscalls

you need to implement these file operation syscalls: **create**, **remove**, **open**, **filesize**, **read**, **write**, **seek**, **tell**, and **close**.

lib/user/syscall.h

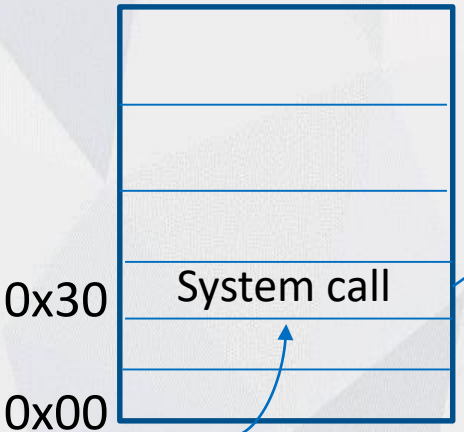
```
bool create (const char *file,
             unsigned initial_size);
bool remove (const char *file);
int open (const char *file);
int filesize (int fd);
int read (int fd, void *buffer, unsigned length);
int write (int fd, const void *buffer,
           unsigned length);
void seek (int fd, unsigned position);
unsigned tell (int fd);
void close (int fd);
```

System Call Overview

1. Pintos using “software interrupt ”(**int $0x30**) to invoke a system call.
2. System call number and any additional arguments are expected to be pushed on the stack .
3. System call handler **syscall_handler()** get control.

User Program

```
main() {
    ……
    exec(file_name);
}
```



userprog/syscall.c

```
syscall_handler(struct
                intr_frame *f)
{
    int call_num =
    (uint32_32*)f->esp;
    if(call_num==SYS_EXEC){
            ……
    }
}
```

lib/user/syscall.c

```
pid_t exec (const char *file){
  return (pid_t) syscall1 (SYS_EXEC,
          file);
}
```

What We Focus in Report

Explain 4 aspects of your report

- Data structure
- Algorithms
- Synchronization (e.g. the filesystem is not thread-safe,how to implement thread-safe)
- Rationale (why this better and how much coding)

Register team

- Since project 2 is a team homework.
- Project 2 is team project, at most two students per group.
- Please use the link below to register the team information before **April 14, 18:00.**

https://docs.qq.com/sheet/DTUhoUkhmR3pYWkFC

Thank you for listening!