POSTECH

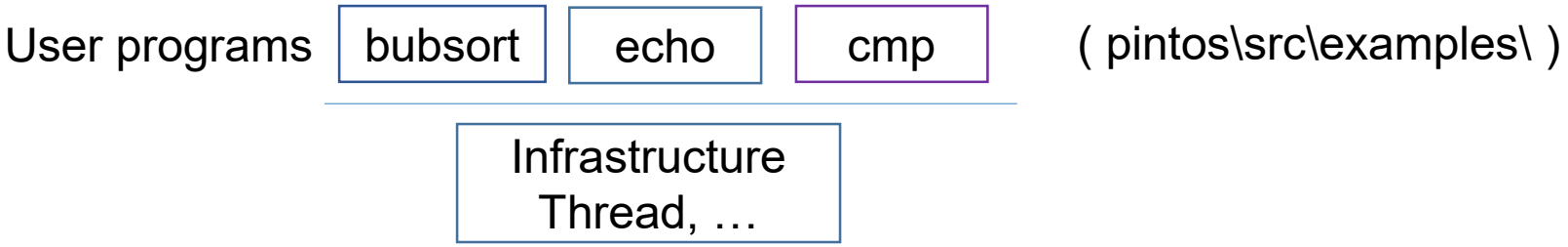
CSED 312:

Operating System Lab

Project2. User Programs

Autumn 2023

Introduction



- Goal
  - Allow running user programs
- Build services for user programs to use
  - Command-line argument passing
  - Process termination messages
  - System calls for
    - User process manipulation : halt(), exit(), exec(), wait()
    - Basic file manipulation : create(), open(), read(), write(), …
  - Write protection on executable files in use

Requirements

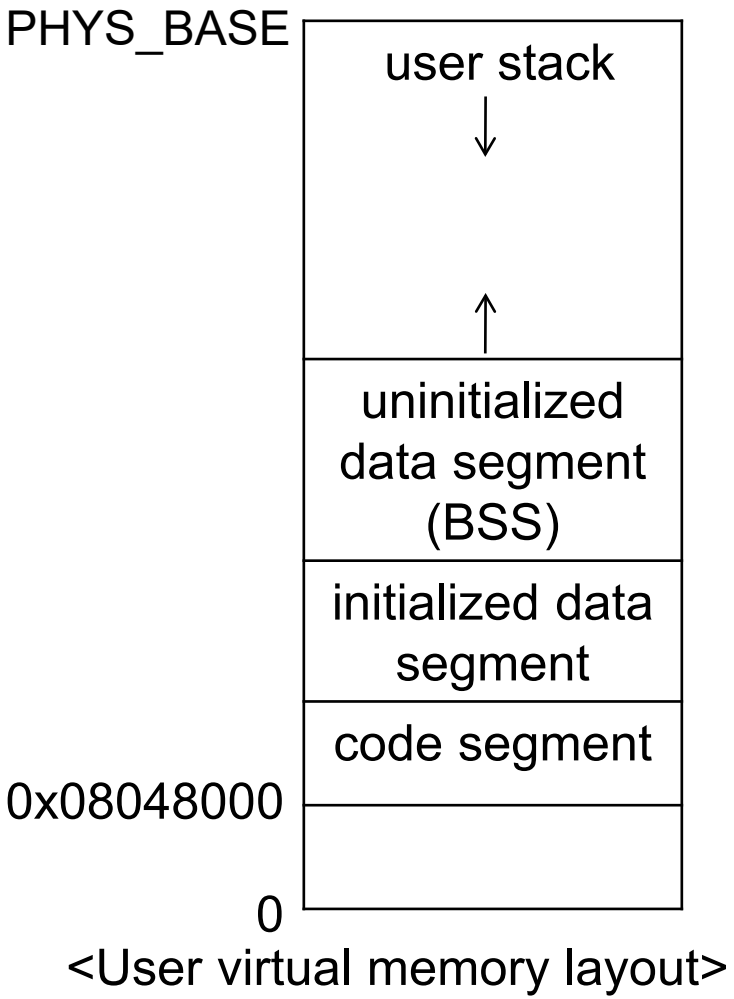
- Process Termination Messages (5 points)
- Argument Passing (5 points)
- System Call (20 points)
- Denying Writes to Executables (5 points)

1. Process Termination Messages (5 Points)

- Process Termination Messages (5 Points)
  - Print the process's name and exit code
    - printf ("%s: exit(%d)\n", *process_name*, *exit_code*);
    - e.g.) args-single: exit(0)
    - Do not print these messages when a kernel thread terminates or the halt system call is invoked
  - Don't print any other additional messages

2. Argument Passing (5 Points) (1/2)

- Implement argument passing
  - current implementation: taking the command line as the program file name
  - extending **process_execute()** to divides the command line into words at spaces
- "ls -l foo bar" on command line :
  - process_execute("ls -l foo bar");
  - run the program file "ls" with three arguments "-l", "foo" and "bar".
  - The caller's stack pointer is accessible as the **esp** member of the struct **intr_frame.** *(threads/interrupt.h)*
- 80x86 convention
  - Arguments are pushed on the stack in **right-to-left** order.
  - The caller pushes the address of its next instruction
  - The callee executes



<User virtual memory layout>

2. Argument Passing (5 Points) (2/2)

- Example: State of the stack at the beginning of the user program (/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 (*) ()

```
bfffffc0                                           00 00 00 00 |            ....|
bfffffd0  04 00 00 00 d8 ff ff bf-ed ff ff bf f5 ff ff bf |................|
bfffffe0  f8 ff ff bf fc ff ff bf-00 00 00 00 00 2f 62 69 |............./bi|
bffffff0  6e 2f 6c 73 00 2d 6c 00-66 6f 6f 00 62 61 72 00 |n/ls.-l.foo.bar.|
```

POSTECH

3. System call (20 Points) (1/2)

- System call: internal interrupts or software exceptions
- Implement the system call handler
  - Implement your code on syscall.c and syscall.h in userprog folder.
  - System call numbers for each system call are defined in lib/syscall-nr.h

3. System call (20 Points) (2/2)

- User Process Manipulation (10 Points)
  - Services that enable user process to control other processes
  - halt(), exit(), exec(), wait()

- File Manipulation (10 Points)
  - Basic file system functions are already implemented (in filesys/filesys.c and file.c)
  - create(), remove(), open(), filesize(), read(), write(), seek(), tell(), close()
  - Provide system calls so that user programs can access on the functions
  - File descriptor
    - Non-negative integer
    - 0 and 1 are reserved for the console
    - Each process has an independent set of file descriptors
    - File descriptors are *not inherited* by child processes
    - A file can have multiple file descriptors

4. Denying Writes to Executables (5 Points)

- Denying Writes to Executables (5 Points)
  - Deny any attempts to write on the program file that is running
    - e.g., while ‘echo’ is running, writing anything on ‘echo’ is not allowed
  - Call file_deny_write() and file_allow_write() at the appropriate moments

Tips

- Use codes in “src/userprog” directory for this project
  - NO code from the project 1 is required for this project
  - You may work with
    - Your Project 1 source code
    - The clean one
  - To run test suites, make in “src/userprog” and make check in “src/userprog/build”
- You need to understand about…
  - Virtual Memory layout in Pintos
    - Its structure, and accessing method
  - Structure of thread and process
    - Relations with parent and child
  - System call handler
  - Basic file system
    - Relationship between file and inode

Tips: File system disk

- “filesys.dsk” : virtual disk file for pintos
- In “userprog/build”,
  - “**pintos-mkdisk filesys.dsk --filesys-size=2**”
    : create 2 MB size disk
  - “**pintos -f -q**”
    : format the disk
  - “**pintos -p *file* -a *newfile* -- -q**”
    : put file into pintos as newfile
    - E.g., pintos -p ../../examples/echo -a echo -- -q
    - *Before that, build examples (type ‘make’ at examples directory)*
  - “**pintos -q run ‘*file arg1 arg2 arg3 …*’**”
  - : execute the program
    - E.g., “pintos -q run ‘echo’”
  - **pintos --filesys-size=2 -p ../../examples/echo -a echo -- -f -q run 'echo x‘**
    - Automatically make filesys.dsk and delete it after execution

Tips: File system disk (Cont.)

- The Pintos automatic test suite creates temporary file system disk for you.

```
deokhk@DESKTOP-1P9U555:~/pintos_pj2/src/userprog/build$ make check
pintos -v -k -T 60 --qemu  --filesys-size=2 -p tests/userprog/args-single -a args-sin
gle -- -q  -f run 'args-single onearg' < /dev/null 2> tests/userprog/args-single.erro
rs > tests/userprog/args-single.output
perl -I../.. ../../tests/userprog/args-single.ck tests/userprog/args-single tests/use
rprog/args-single.result
pass tests/userprog/args-single
```

Provided resources

- project2.pdf
- project2_requirements.docx
  - Essential requirements and explanation
- pintos.pdf
  - Official manual for the pintos project
    - Detailed description of the concept covered in this project
    - What needs to be considered when you implement each of the requirements
  - We highly recommend you read relevant chapters before you start the project
  - It will save you lots of times

Design report should include

- How to achieve each requirements
  - Big picture: how to solve problems
  - Data structure and detailed algorithm
- Analysis on process execution procedure
  - explain the procedure of process execution in the current pintos system
  - see source codes (“threads/init.c”, “userprog/process.c”)
- Analysis on system call procedure
  - explain how to call syscall_handler() in userprog/syscall.c from user program
  - see source codes (“lib/user/syscall.c”, “threads/intr-stubs.S”, “threads/interrupt.c”) and section 3.5.2 on pintos manual
- Analysis on file system
  - structure(file, inode), functions(need to implement system call) of the file system in pintos
  - see source codes (“filesys/ file.c”, “filesys/ inode.c” “filesys/filesys.c”) and section 3.1.2 on the manual

Submitting Project2

- Server information
  - Server IP: 141.223.121.130
  - Same server we used for project1 submission (changed server)
- Whole project source code must be submitted to server
  - Submit your entire project files at "/home/teamXX/pintos" (XX is your team ID)
  - Must include ".git" folder in the project files.
  - Ex) git clone "your git source" pintos
- Git branch naming
  - Project implementation must be submitted under **"project2" branch**
  - Make sure all your features are merged in "project2" before submission.
- Submission due
  - ~ 2023.11.07 17:59:59

Announcements

- Project 1 Demo is today
  - A – Hogil Kim Building 303; B – Hogil Kim Building 304; C – Hogil Kim Building 305.
- Project 2 Demo/ Quiz & Project 3 announcement on 11/7 (Tues.)
  - A few questions about pintos project and source codes
- Q&A
  - You can use Q&A in PLMS (general rule)
  - Or email me (for private questions only): deokhk@postech.ac.kr
- We will use PLMS for any additional announcements