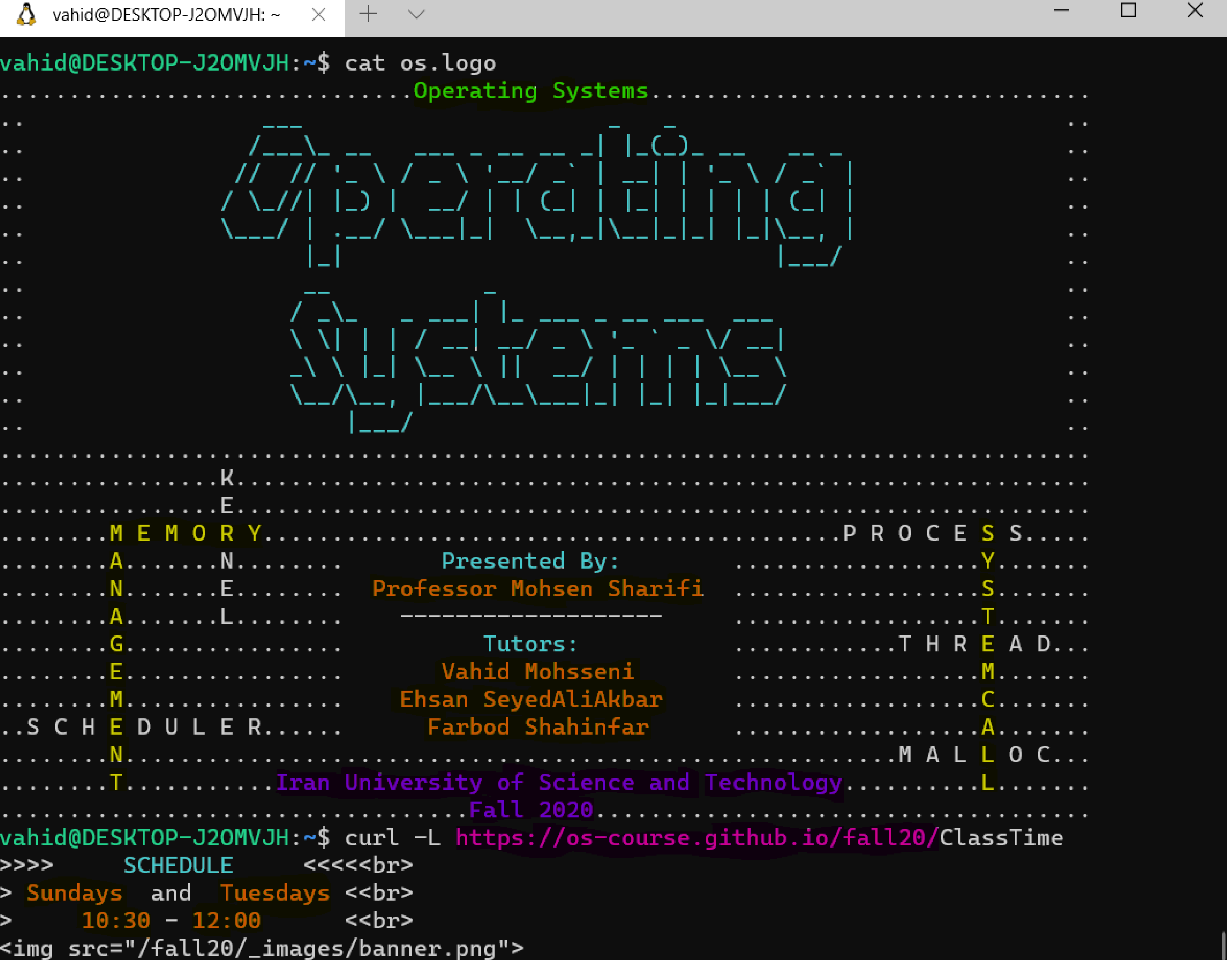
Operating Systems

Process and OS

Fall 2020


vahid@DESKTOP-J2OMVJH: ~
vahid@DESKTOP-J2OMVJH:~$ cat os.logo
Operating Systems
KERNEL
MEMORY MANAGEMENT
PROCESS
SYSTEM CALL
THREAD
SCHEDULER
MALLOC
Presented By:
Professor Mohsen Sharifi
Tutors:
Vahid Mohsseni
Ehsan SeyedAliAkbar
Farbod Shahinfar
Iran University of Science and Technology
Fall 2020
vahid@DESKTOP-J2OMVJH:~$ curl -L https://os-course.github.io/fall20/ClassTime
>>>> SCHEDULE <<<<
> Sundays and Tuesdays <
> 10:30 - 12:00 <
<img src="/fall20/_images/banner.png">

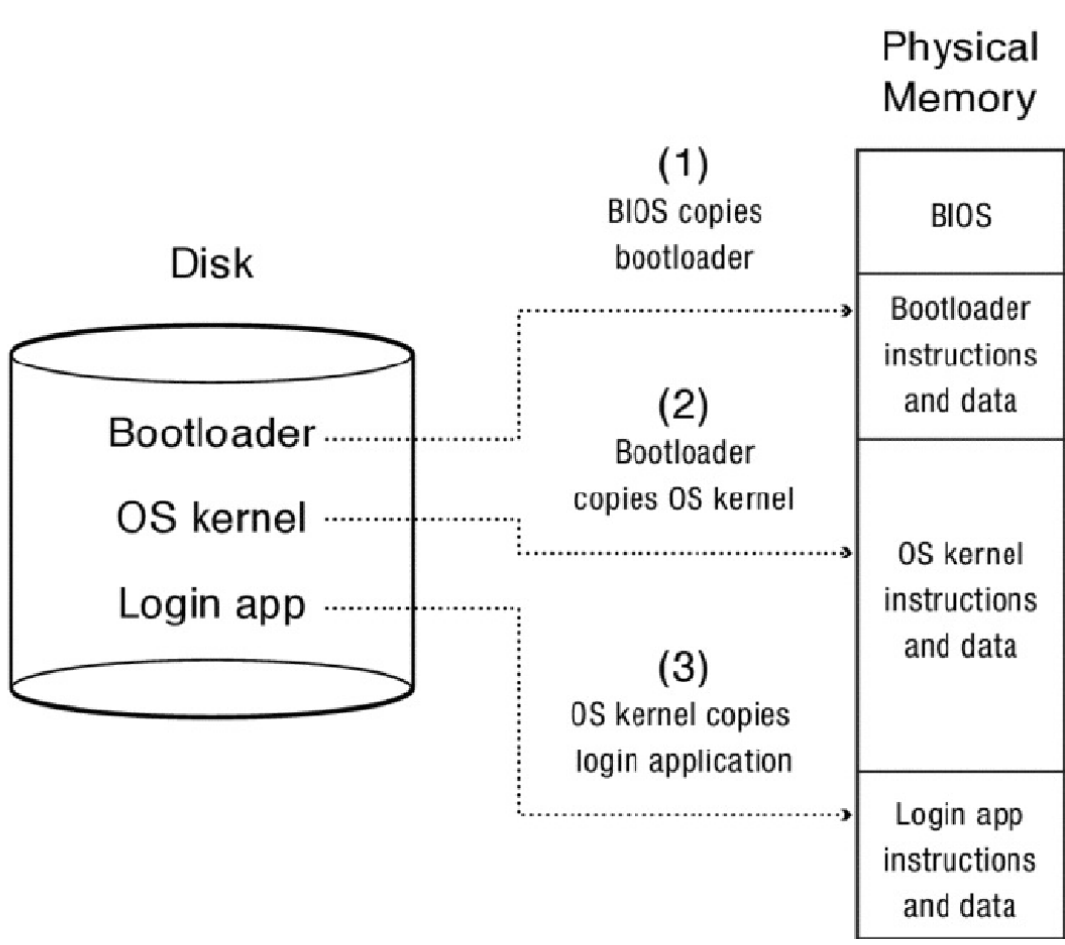
Groups

- To Students whom were absent in meeting:
- I am available at 13 to 14 today. The link will be shared in the group.
- Please join and then we can meet!

Quiz

- Quiz is after class.
- 11:50 to 12:30.
- 10 minutes time to answer.

So far …



Process: an Abstraction

- Informally, a running program.
- Is a lifeless thing: a bunch of instructions (maybe data) on the disk
- Waiting to be brought into action.
- Ready, Running, Waiting.

OS

- Load instruction and data segments of executable file into memory
- Create stack and heap
- Transfer control to program
- Provide services to program
- While protecting everything

Heap vs. Stack

- Will be discussed in Memory Management session.

```
#include <stdio.h>


double multiplyByTwo (double input) {
  double twice = input * 2.0;
  return twice;
}


int main (int argc, char *argv[])
{
  int age = 30;
  double salary = 12345.67;
  double myList[3] = {1.2, 2.3, 3.4};


  printf("double your salary is %.3f\n",
multiplyByTwo(salary));


  return 0;
}
```

Heap vs. Stack

```
#include <stdio.h>


double multiplyByTwo (double input) {

  double twice = input * 2.0;

  return twice;

}


int main (int argc, char *argv[])

{

  int age = 30;

  double salary = 12345.67;
```

```
  double myList[3] = {1.2, 2.3, 3.4};


  printf("double your salary is %.3f\n",
multiplyByTwo(salary));


  return 0;

}
```

STACK

Heap vs. Stack

```
#include <stdio.h>

#include <stdlib.h>


double *multiplyByTwo (double *input) {

  double *twice = malloc(sizeof(double));

  *twice = *input * 2.0;

  return twice;

}


int main (int argc, char *argv[])

{

  int *age = malloc(sizeof(int));

  *age = 30;

  double *salary = malloc(sizeof(double));

  *salary = 12345.67;

  double *myList = malloc(3 * sizeof(double));

  myList[0] = 1.2;

  myList[1] = 2.3;

  myList[2] = 3.4;


  double *twiceSalary = multiplyByTwo(salary);


  printf("double your salary is %.3f\n", *twiceSalary);


  free(age);

  free(salary);

  free(myList);

  free(twiceSalary);


  return 0;

}
```

Heap vs. Stack

```
#include <stdio.h>
#include <stdlib.h>

double *multiplyByTwo (double *input) {
  double *twice = malloc(sizeof(double));
  *twice = *input * 2.0;
  return twice;
}

int main (int argc, char *argv[])
{
  int *age = malloc(sizeof(int));
  *age = 30;
  double *salary = malloc(sizeof(double));
  *salary = 12345.67;
  double *myList = malloc(3 * sizeof(double));
  myList[0] = 1.2;
  myList[1] = 2.3;
  myList[2] = 3.4;

  double *twiceSalary = multiplyByTwo(salary);

  printf("double your salary is %.3f\n", *twiceSalary);

  free(age);
  free(salary);
  free(myList);
  free(twiceSalary);

  return 0;
}
```

HEAP

Heap vs. Stack

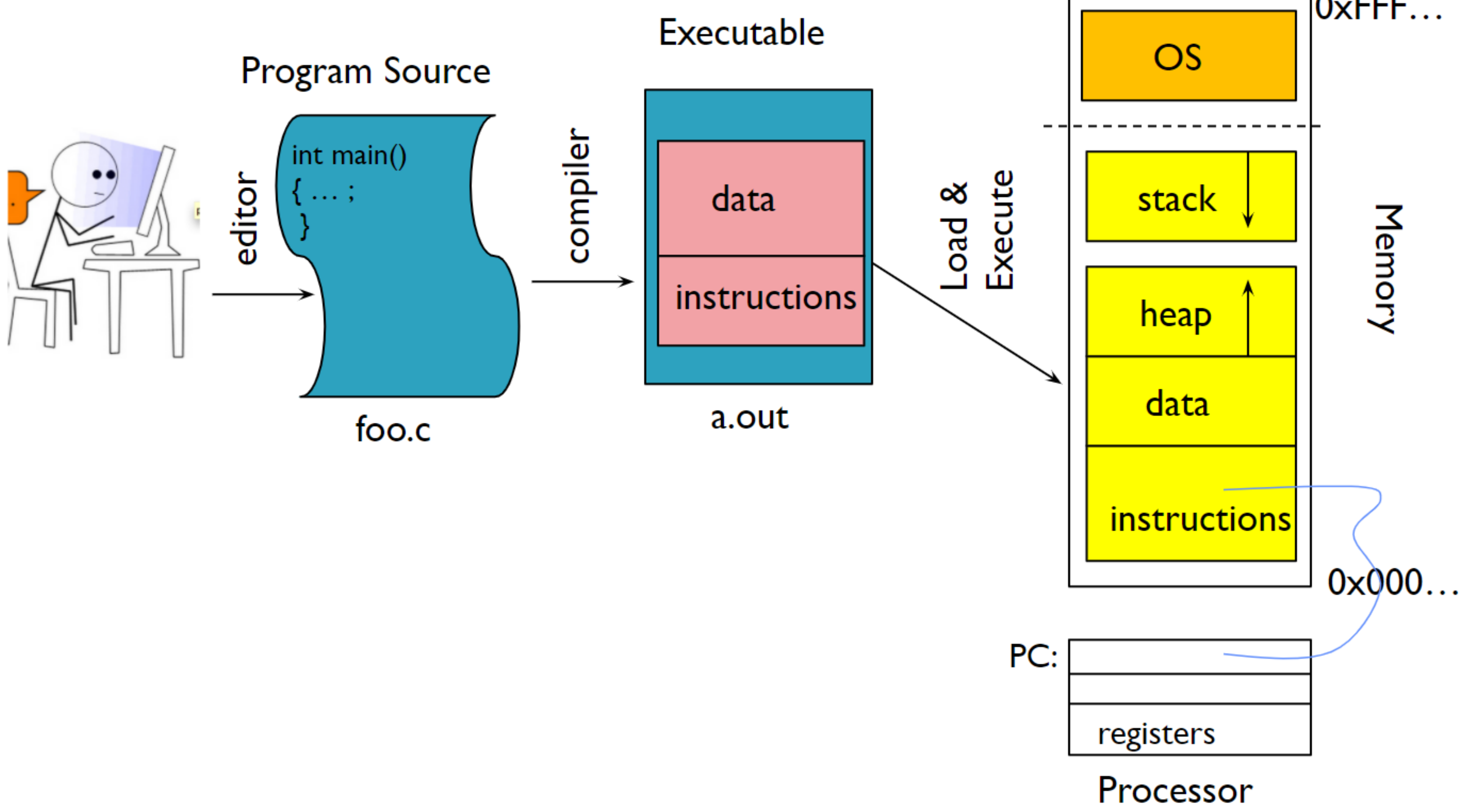
Stack

- don't have to explicitly de-allocate variables
- space is managed efficiently by CPU, memory will not become fragmented
- local variables only
- limit on stack size (OS-dependent)
- variables cannot be resized

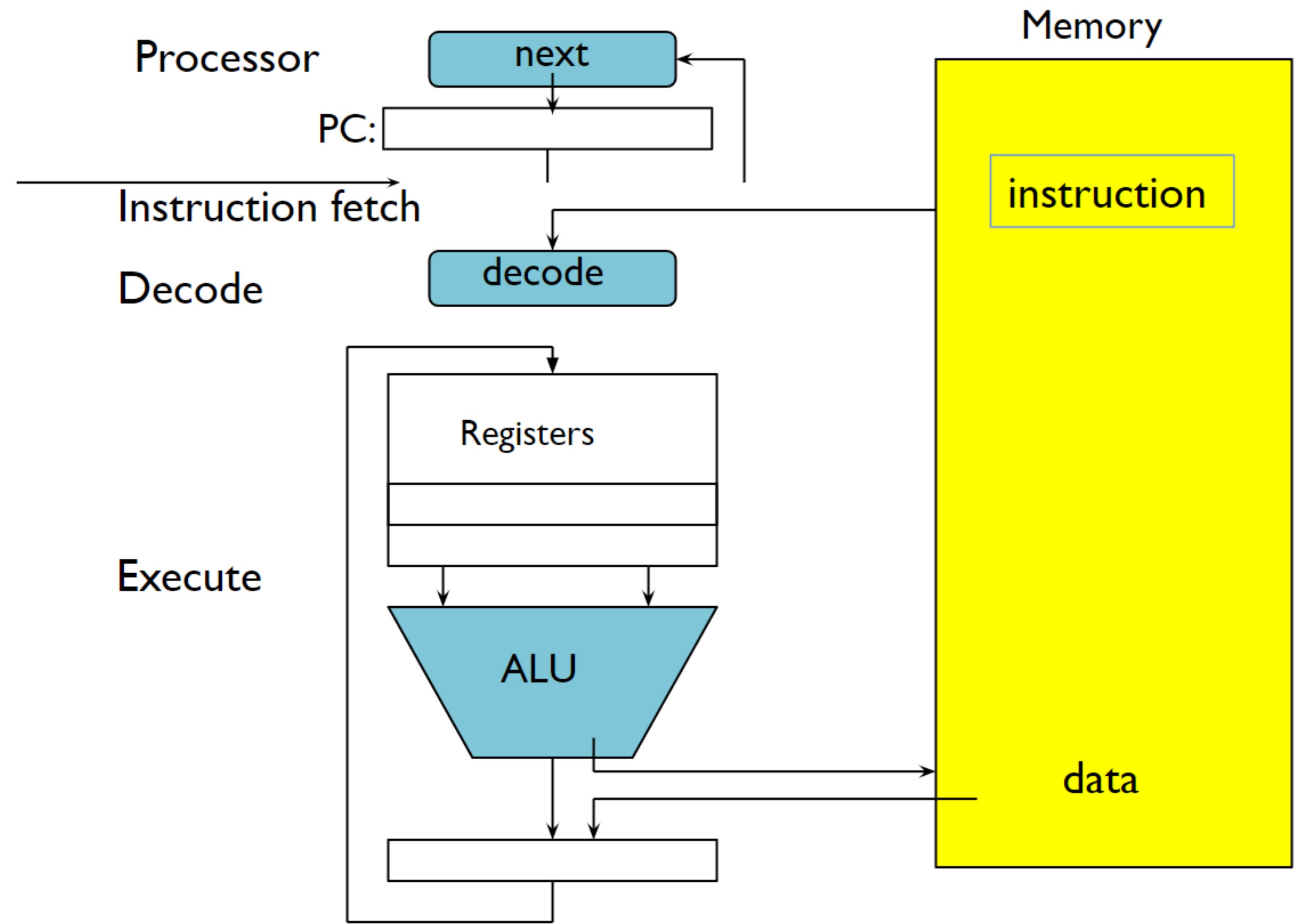
Heap

- variables can be accessed globally
- no limit on memory size
- no guaranteed efficient use of space, memory may become fragmented over time as blocks of memory are allocated, then freed
- you must manage memory (you're in charge of allocating and freeing variables)
- variables can be resized using realloc()

Run Program: 1



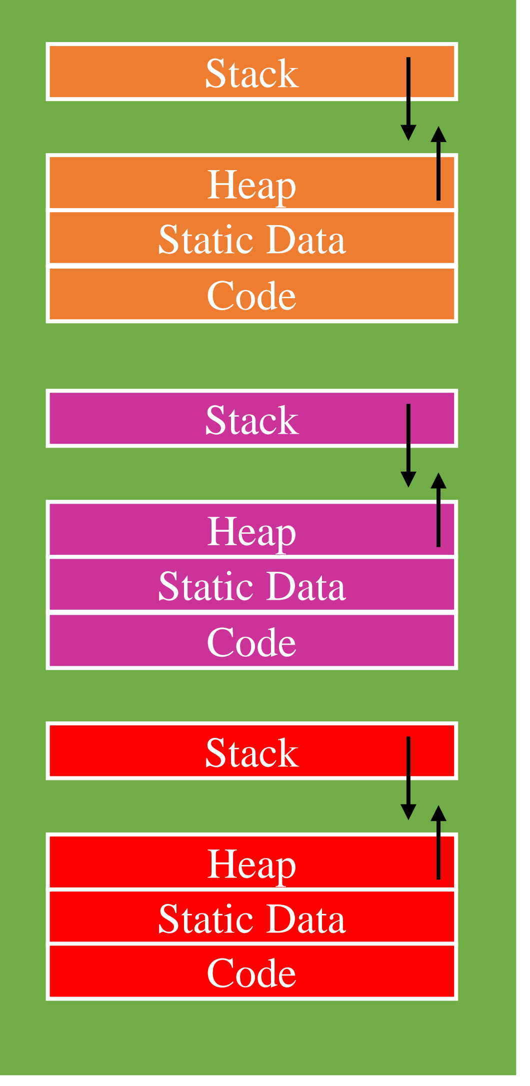
Run Program: 2 Fetch/Decode/Execute



Run Program: 3 Steps

- Execution sequence:
  - Fetch Instruction at PC
  - Decode
  - Execute (possibly using registers)
  - Write results to registers/mem
  - PC = Next Instruction(PC)
  - Repeat

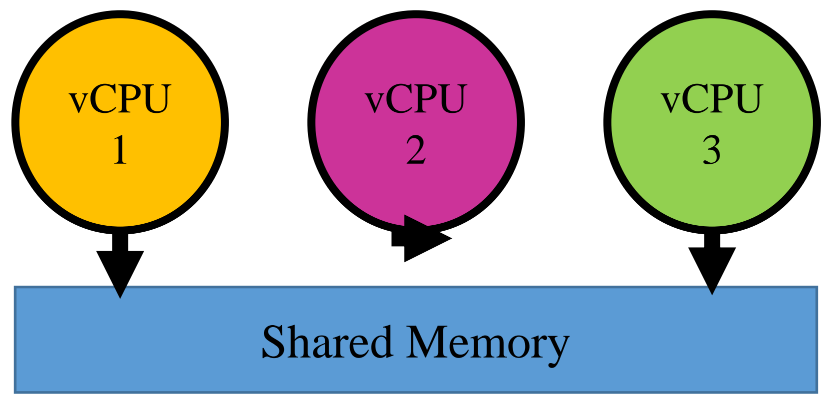
Context-Switching

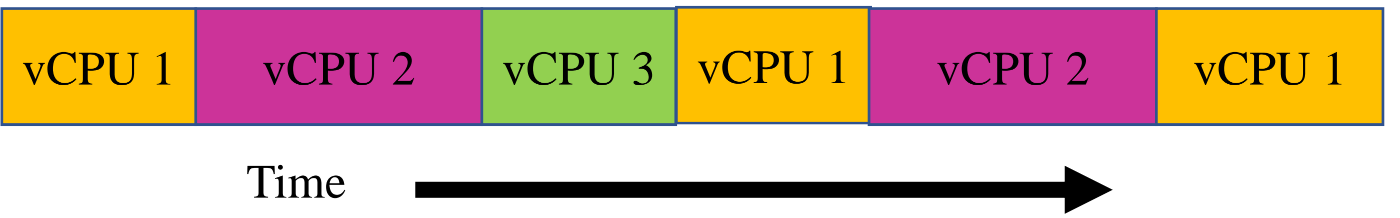


Illusion of Multiple Processors

- Assume a single processor. How do we provide the illusion of multiple processors?
  - Multiplex in time!
- Each virtual “CPU” needs a structure to hold:
  - Program Counter (PC), Stack Pointer (SP)
  - Registers (Integer, Floating point, others…?)
- How switch from one virtual CPU to the next?
  - Save PC, SP, and registers in current state block
  - Load PC, SP, and registers from new state block
- What triggers switch?
  - Timer, voluntary yield, I/O, other things

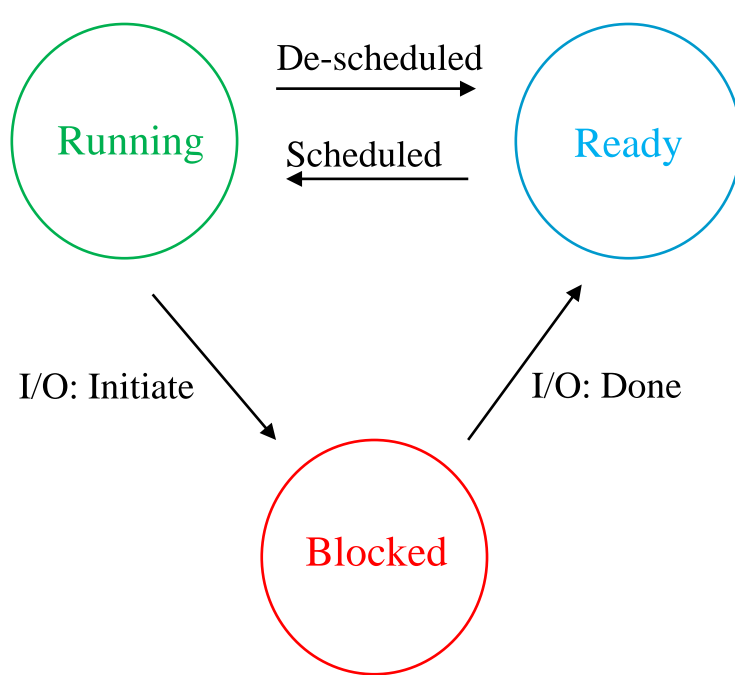
Illusion of Multiple Processors





State Transitions

- Running: A process is running on a processor. It is executing the instructions.
- Ready: A process is ready to run but for some reason OS has chosen not to run it in this moment.
- Blocked: A process has requested some kind of operations (e.g. I/O) that makes it not ready to run until some other events take place.



Process Structure

```
// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE };

// the information xv6 tracks about each process
// including its register context and state
struct proc {
  char *mem;                    // Start of process memory
  uint sz;                      // Size of process memory
  char *kstack;                 // Bottom of kernel stack
                                // for this process
  enum proc_state state;        // Process state
  int pid;                      // Process ID
  struct proc *parent;          // Parent process
  void *chan;                   // If non-zero, sleeping on chan
  int killed;                   // If non-zero, have been killed
  struct file *ofile[NOFILE]; // Open files
  struct inode *cwd;            // Current directory
  struct context context;       // Switch here to run process
  struct trapframe *tf;         // Trap frame for the
                                // current interrupt
};
```

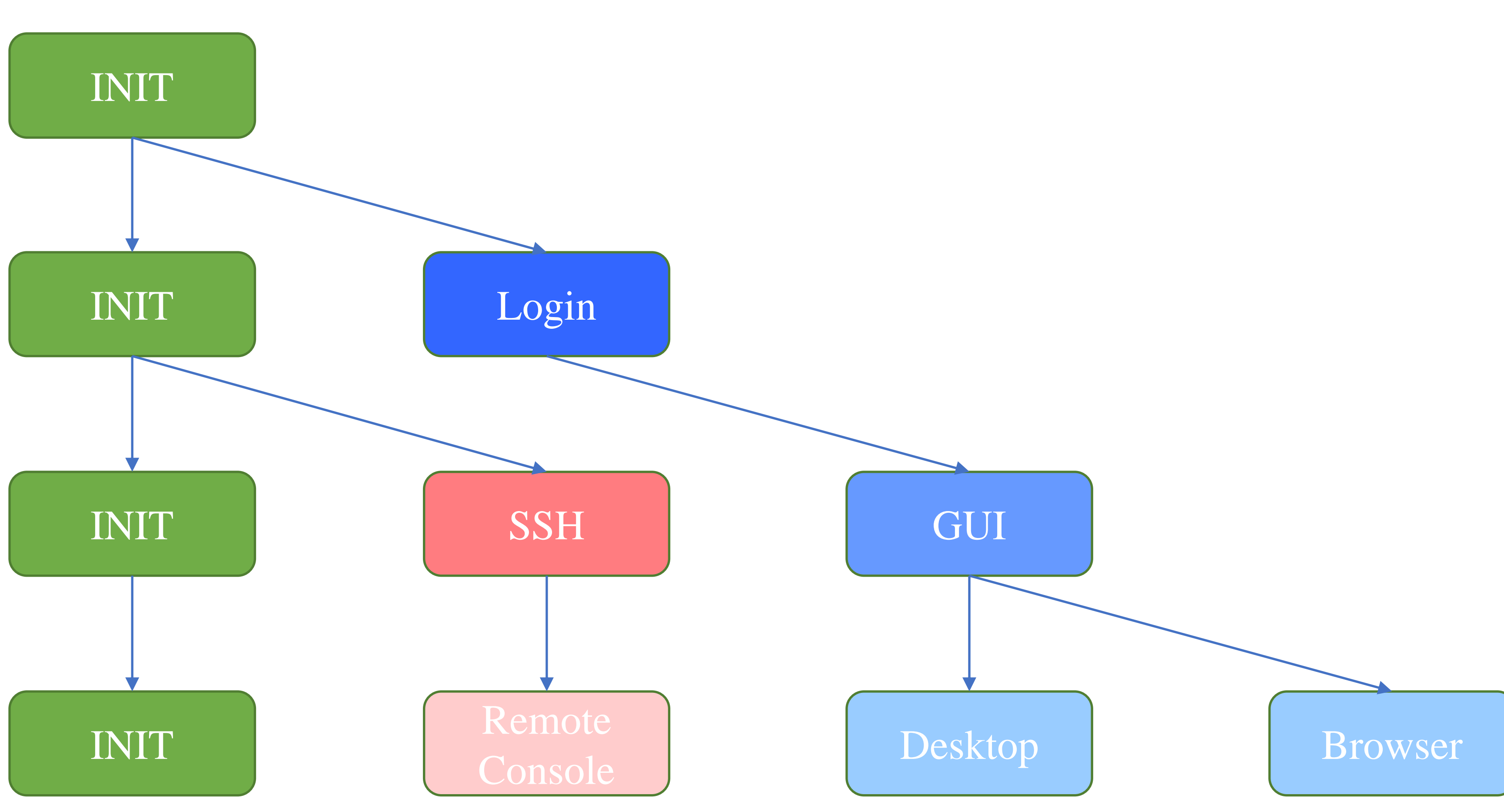
It is Unique for each proc

Process struct in Xv6

Process API

- Create
- Destroy
- Wait
- Other controls
- Status

How to create?



Fork!

- `Fork` is a system-call for creating new process.
- Exact copy of current process with different PID.
- Returns an integer:
  - > 0: running in the context of (original process) parent.
  - = 0: running in the context of (new process) child.
  - < 0: Error! running in the context of original process.

Fork!

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>



int main (int argc, char *argv[])
{
    printf("Hello World! My pid is: %d\n", getpid());

    int r = fork();

    if (r < 0){
        printf("fork failed!\n");
        exit(1);
    } else if (r == 0){
        // Child Process
        printf("Hello World, I am child process. My pid is: %d\n", getpid());

    } else{
        // Parent Process
        printf("I am parent of %d. pid is: %d\n", r, getpid());
    }

    return 0;
}
```

```
Hello World! My pid is: 389
I am parent of 390. pid is: 389
Hello World, I am child process. My pid is: 390
```

Fork and Wait

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>



int main (int argc, char *argv[])
{
    printf("Hello World! My pid is: %d\n", getpid());

    int r = fork();

    if (r < 0){
        printf("fork failed!\n");
        exit(1);
    } else if (r == 0){
        // Child Process
        printf("Hello World, I am child process. My pid is: %d\n", getpid());

    } else{
        // Parent Process
        int w = wait(NULL);
        printf("I am parent of %d. pid is: %d\n", r, getpid());
    }

    return 0;
}
```

```
Hello World! My pid is: 494
Hello World, I am child process. My pid is: 495
I am parent of 495. pid is: 494
```

Fork and Wait and Exec

```
} else if (r == 0){
    // Child Process
    printf("Hello World, I am child process. My pid is: %d\n", getpid());
    char *my_args[3];
    my_args[0] = strdup("./script.sh");
    my_args[1] = strdup("Hello from Bash!");
    my_args[2] = NULL;       // Indicating end of array
    execvp(my_args[0], my_args);

    // CHILD process will be terminated before this line!
    printf("This should not be printed");
```

```
Hello World! My pid is: 102
Hello World, I am child process. My pid is: 103
running bash...
Hello from Bash!
I am parent of 103. pid is: 102
```

Process Management

- `ps aux | grep  process_name`
- `ps -p process_id`
- `pstree      // list tree view of processes`
- `ls -la /proc/3956/`

Process Management

- `top`

Updates frequently the information of running processes.

Poor Processes

Zombie

Orphan

Orphan

- A process whose parent process no more exists Fetch Instruction at PC
- Either finished or terminated without waiting for its child
- The orphan process is soon adopted by `init` process, once its parent process dies.

Orphan

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    // Create a child process
    int pid = fork();

    if (pid > 0)
        printf("in parent process");

    // Note that pid is 0 in child process
    // and negative if fork() fails
    else if (pid == 0)
    {
        sleep(30);
        printf("in child process");
    }

    return 0;
}
```

Zombie

- When a process ends, all of the memory and resources associated with it are deallocated so they can be used by other processes.
- However, the process's entry in the process table remains.
- The zombie processes can be removed from the system by sending the SIGCHLD signal to the parent, using the kill command. If the zombie process is still not eliminated from the process table by the parent process, then the parent process is terminated if that is acceptable.
- The zombie's process ID and entry in the process table can then be reused.
- However, if a parent ignores the SIGCHLD, the zombie will be left in the process table.

Zombie

```c
// A C program to demonstrate Zombie Process.
// Child becomes Zombie as parent is sleeping
// when child process exits.

#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // Fork returns process id
    // in parent process
    pid_t child_pid = fork();

    // Parent process
    if (child_pid > 0)
        sleep(50);

    // Child process
    else
        exit(0);

    return 0;
}
```

What does a zombie look like?

- normal (no zombie)

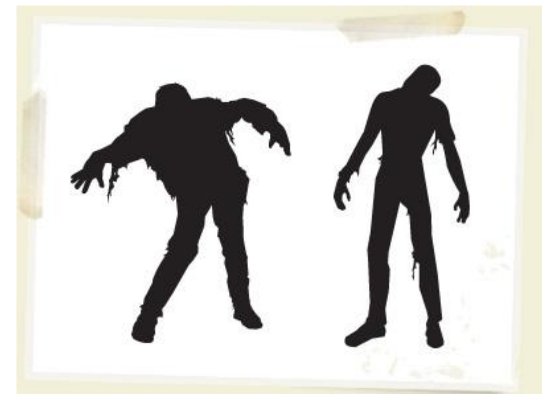
```
$ ps
```

PID	TTY	TIME	CMD
1074	pts/2	00:00:00	bash
1280	pts/2	00:00:00	parentTest.exe
1281	pts/2	00:00:00	childTest.exe
1283	pts/2	00:00:00	ps

- abnormal (zombie)

```
$ ps
```

PID	TTY	TIME	CMD
1074	pts/2	00:00:00	bash
1280	pts/2	00:00:00	parentTest.exe
1281	pts/2	00:00:00	childTest.exe <defunct>
1288	pts/2	00:00:00	ps



What does a zombie look like?

```
$ ps -l

  F S   UID   PID  PPID  C PRI  NI ADDR    SZ  WCHAN TTY          TIME CMD
000 S   561  1074  1073  0  76   0    -   628 11a418 pts/2    00:00:00 bash
000 S   561  1301  1074  0  70   0    -   436 11f22b pts/2    00:00:00 parentTes
004 Z   561  1302  1301  0  70   0    -     0 119ffb pts/2    00:00:00 childTest
000 R   561  1320  1074  0  77   0    -   646      - pts/2    00:00:00 ps
```

Questions?