

R

بسم الله الرحمن الرحيم

# سیستم عامل

جلسه چهارم – ادامه‌ی پردازش، ریسمان

یادآوری!

■ دوشنبه آزمونک اول است!

# آنچه گذشت

جلسه‌ی قبل، پردازش

# The Process Concept (Vs. Program)

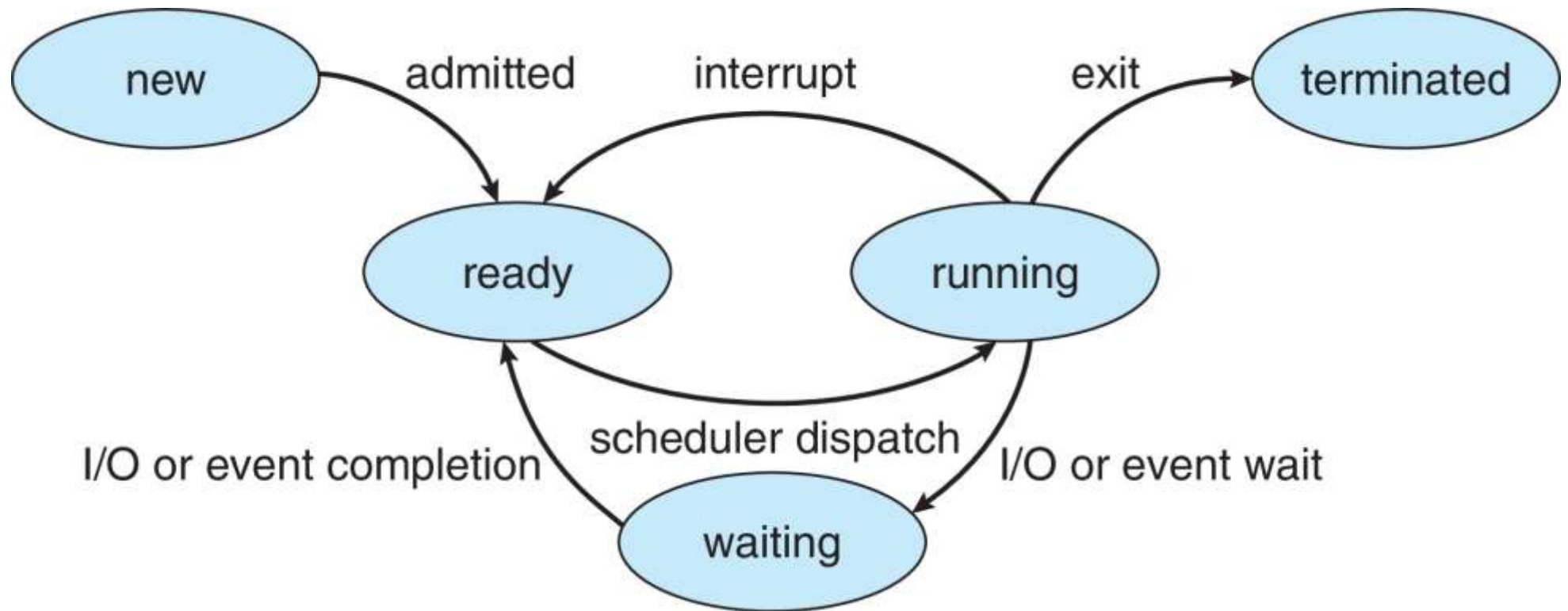
- Process – a program in execution
- Program
  - description of how to perform an activity
  - instructions and static data values
- Process
  - a snapshot of a program in execution
  - memory (program instructions, static and dynamic data values)
  - CPU state (registers, PC, SP, etc)
  - operating system state (open files, accounting statistics etc)

# Process Control Block (PCB)

- **Process state** – running, waiting, etc.
- **Program counter** – location of instruction to next execute
- **CPU registers** – contents of all process-centric registers
- **CPU scheduling information**- priorities, scheduling queue pointers
- **Memory-management information** – memory allocated to the process
- **Accounting information** – CPU used, clock time elapsed since start, time limits
- **I/O status information** – I/O devices allocated to process, list of open files



# Process State in More Detail

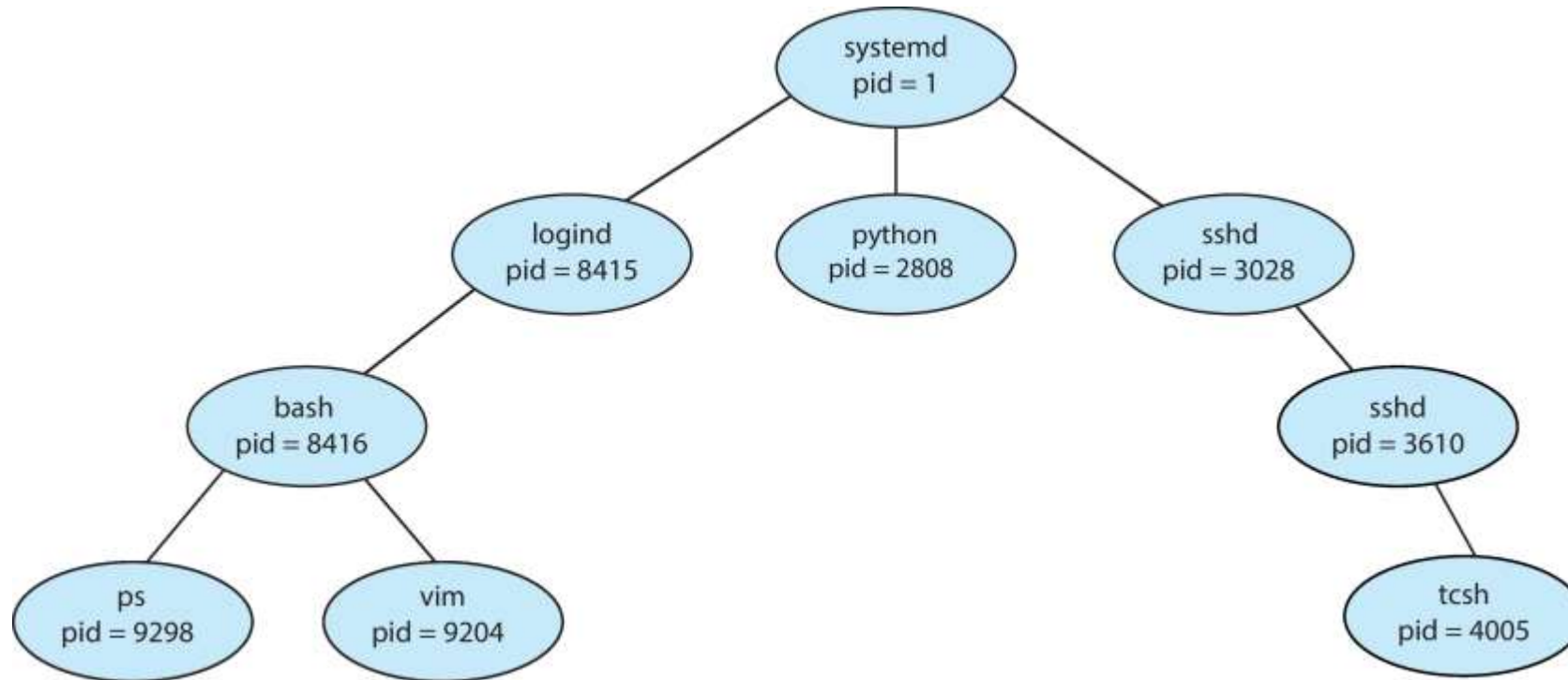


# Question

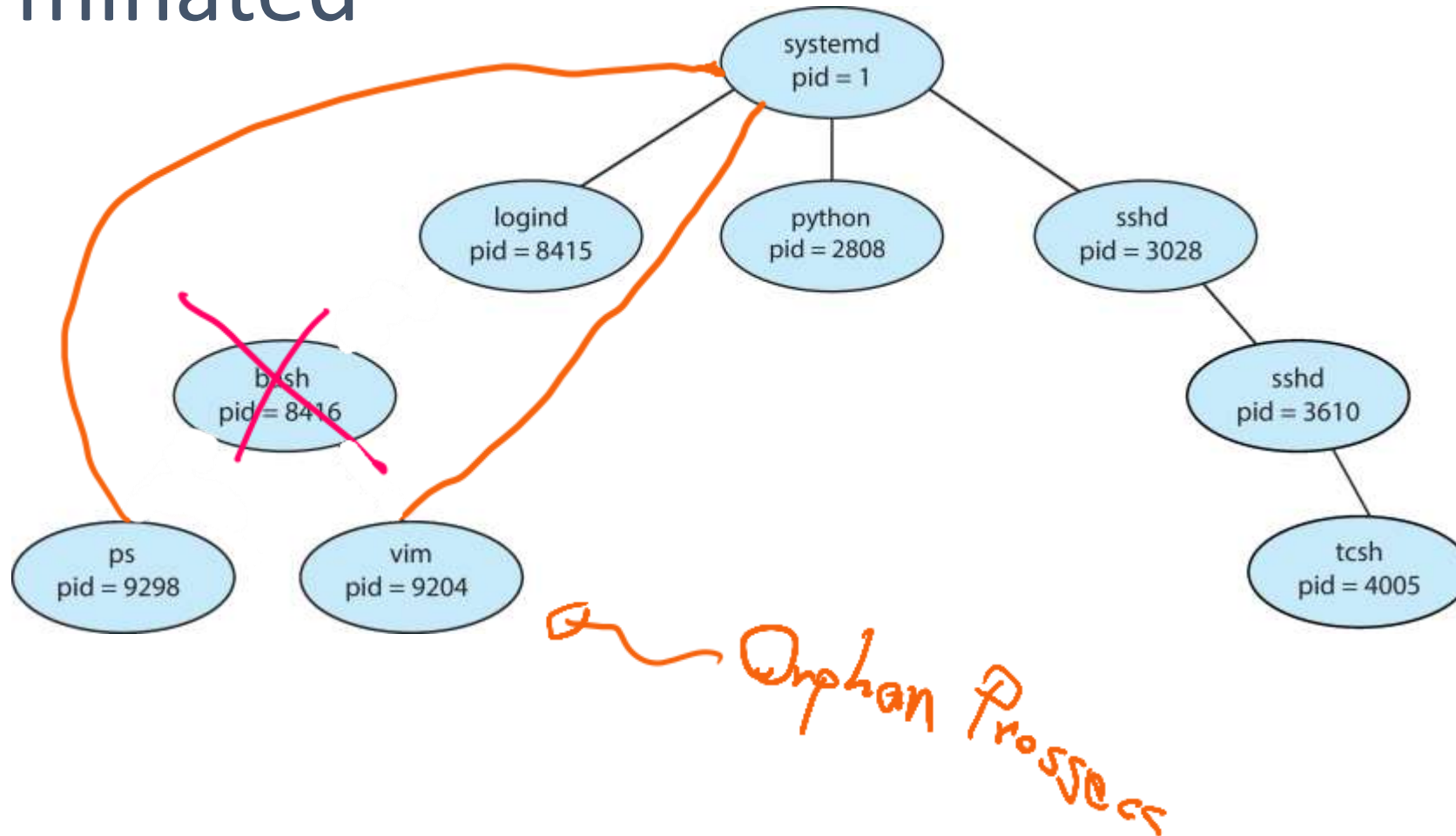
- Why there are new and terminated states ?
- New State:
  - *Prevents the system from scheduling an incomplete or uninitialized process.*
- Terminated State:
  - *Ensures that resources used by the process are properly cleaned up and that exit statuses are correctly handled.*
- Zombie process:
  - *The parent wait() to collect exit status*
  - *Child process in terminated states (Zombie)*



# A Tree of Processes in Linux



# What Happen if a parent process terminated



# Process Creation in UNIX

- All processes have a unique process id

- ❖ ***getpid()**, **getppid()** system calls allow processes to get their information*

- Process creation

- ❖ ***fork()** system call creates a copy of a process and returns in both processes (parent and child), but with a different return value*
- ❖ ***exec()** replaces an address space with a new program*

- Process termination, signaling

- ❖ ***signal()**, **kill()** system calls allow a process to be terminated or have specific signals sent to it*

# Process Creation Example

csh (pid = 22)

```
...  
  
pid = fork()  
if (pid == 0) {  
    // child...  
    ...  
    exec("/bin/ls");  
}  
else {  
    // parent  
    wait();  
}  
...
```

# Process Creation Example

csch (pid = 22)

```
...  
  
pid = fork()  
if (pid == 0) {  
    // child...  
    ...  
    exec("/bin/ls");  
}  
else {  
    // parent  
    wait();  
}  
...
```

csch (pid = 24)

```
...  
  
pid = fork()  
if (pid == 0) {  
    // child...  
    ...  
    exec("/bin/ls");  
}  
else {  
    // parent  
    wait();  
}  
...
```

# Process Creation Example

csh (pid = 22)

```
...  
  
pid = fork()  
if (pid == 0) {  
    // child...  
    ...  
    exec("/bin/ls");  
}  
else {  
    // parent  
    wait();  
}  
...
```

csh (pid = 24)

```
...  
  
pid = fork()  
if (pid == 0) {  
    // child...  
    ...  
    exec("/bin/ls");  
}  
else {  
    // parent  
    wait();  
}  
...
```

# Process Creation Example

csch (pid = 22)

```
...  
  
pid = fork()  
if (pid == 0) {  
    // child...  
    ...  
    exec("/bin/ls");  
}  
else {  
    // parent  
    wait();  
}  
...
```

csch (pid = 24)

```
...  
  
pid = fork()  
if (pid == 0) {  
    // child...  
    ...  
    exec("/bin/ls");  
}  
else {  
    // parent  
    wait();  
}  
...
```

# Process Creation Example

csch (pid = 22)

```
...  
  
pid = fork()  
if (pid == 0) {  
    // child...  
    ...  
    exec("/bin/ls");  
}  
else {  
    // parent  
    wait();  
}  
...
```

ls (pid = 24)

```
//ls program  
main() {  
    //look up dir  
    ...  
}
```



# Process Creation (fork)

- Fork creates a new process by *copying* the calling process
- The new process has its own
  - *Memory address space (copied from parent)*
    - Instructions (same program as parent!)
    - Data
    - Stack
  - *Register set (copied from parent)*
  - *Process table entry in the OS*

# Fork Challenge!

What is the output of the program?

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

int main() {
    printf("Parent process started with PID:
%d\n", getpid());

    fork();
    fork();
    printf("Process with PID: %d, Parent PID:
%d\n", getpid(), getppid());

    return 0;
}
```

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

2

Parent process started with PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

4

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2



```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

4

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10

```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID:
%d\n", getpid(), getppid());
return 0;
```

2

```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID: %d\n",
getpid(), getppid());
return 0;
```

4

```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID: %d\n",
getpid(), getppid());
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());  
return 0;
```

4

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

4

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID:
%d\n", getpid(), getppid());
return 0;
```

2

```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID: %d\n",
getpid(), getppid());
return 0;
```

3

```
printf("Parent process started with PID: %d\n", getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID: %d\n", getpid(),
getppid());
return 0;
```

4

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

```
printf("Parent process started with PID: %d\n", getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n", getpid(),  
getppid());  
return 0;
```

4

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n", getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n", getpid(),  
getppid());  
return 0;
```

1

```
printf("Parent process started with PID: %d\n", getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n", getpid(),  
getppid());  
return 0;
```

4

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

```
printf("Parent process started with PID: %d\n", getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n", getpid(),  
getppid());  
return 0;
```

1

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2



```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

6

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

6

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

6!

or

5?

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2

```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID: %d\n",
getpid(), getppid());
return 0;
```

6

```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID: %d\n", getpid(), getppid());
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2  
Process with PID: 3, Parent PID: 1

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

6

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2

Process with PID: 2, Parent PID: 10

Process with PID: 4, Parent PID: 2

Process with PID: 3, Parent PID: 1



```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID: %d\n",
getpid(), getppid());
return 0;
```

6

```
printf("Parent process started with PID: %d\n",
getpid());
fork();
fork();
printf("Process with PID: %d, Parent PID: %d\n", getpid(), getppid());
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2  
Process with PID: 3, Parent PID: 1

```
printf("Parent process started with PID: %d\n",  
getpid());
```

```
fork();
```

```
fork();
```

```
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());
```

```
return 0;
```

6

```
printf("Parent process started with PID: %d\n",  
getpid());
```

```
fork();
```

```
fork();
```

```
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());
```

```
return 0;
```

3

Parent process started with PID: 2

Process with PID: 2, Parent PID: 10

Process with PID: 4, Parent PID: 2

Process with PID: 3, Parent PID: 1

Process with PID 6, Parent PID: 3



```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

6

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2  
Process with PID: 3, Parent PID: 1  
Process with PID 6, Parent PID: 3

```
printf("Parent process started with PID: %d\n", getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n", getpid(),  
getppid());  
return 0;
```

6

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID:  
%d\n", getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2  
Process with PID: 3, Parent PID: 1  
Process with PID 6, Parent PID: 3

```
printf("Parent process started with PID: %d\n", getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n", getpid(),  
getppid());  
return 0;
```

6

```
printf("Parent process started with PID: %d\n",  
getpid());  
fork();  
fork();  
printf("Process with PID: %d, Parent PID: %d\n",  
getpid(), getppid());  
return 0;
```

3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2  
Process with PID: 3, Parent PID: 1  
Process with PID 6, Parent PID: 3

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2  
Process with PID: 3, Parent PID: 1  
Process with PID 6, Parent PID: 3

# Is it the only feasible output?

Parent process started with PID: 2  
Process with PID: 2, Parent PID: 10  
Process with PID: 4, Parent PID: 2  
Process with PID: 3, Parent PID: 1  
Process with PID 6, Parent PID: 3

# Combinatorial Problem

- How many ways that the code can run?

# Questions?

# جلسه‌ی جدید

ادامه‌ی پردازه، ریسمان



تمام شدن یک پروژه

# How Do Processes Terminate?

Conditions that terminate processes:

- *Normal exit (voluntary)*
- *Error exit (voluntary)*
- *Fatal error (involuntary)*
- *Killed by another process (involuntary)*

# Killing a process

- Sending kill signal to kernel
- Killing a process does not kill its descendants

# wait()

- Waits until:
  - *A child is killed/terminated, or*
  - *A signal is received from OS (Interrupt the wait)*

# Some important signals in Linux

- **SIGINT** (Interrupt): Sent when the user interrupts a process (usually with Ctrl+C).
- **SIGKILL** (Kill): Immediately terminates the process. Cannot be ignored or handled by the process.
- **SIGTERM** (Terminate): Requests the process to gracefully terminate. Can be caught to allow cleanup before exiting.
- **SIGSTOP** (Stop): Stops a process execution. Can be resumed later with **SIGCONT**.

# پردازها و همزمانی

■ مثال:

- فرض کنید یک گراف داریم و می‌خواهیم به طور همزمان الگوریتمی مثل DFS را از رئوس آن اجرا کنیم.

■ برنامه‌ی آن چه شکلی است؟

# پردازه‌ها و همزمانی - کد نمونه

Load graph and edges

let U be the set of start vertices for DFS

```
for (int v in U) {  
    Int f = fork();  
    If (fork() == 0) { // child process  
        DFS(v);  
        return 0;  
    }  
}  
  
return 0;
```

# پردازه‌ها و هم‌زمانی - کد نمونه

```
Load graph and edges
let U be the set of start vertices for DFS
for(int v in U):
    Int f = fork()
    If (fork() == 0) { // child process
        DFS(v)
        return 0;
    }
}
return 0;
```

- میزان حافظه‌ی مصرفی با فرض اینکه حجم گراف ۱ گیگابایت باشد؟
- پردازه‌های فرزند، چطوری نتیجه‌ی اجرا را به پردازهی والد بدهند؟



رېسمان

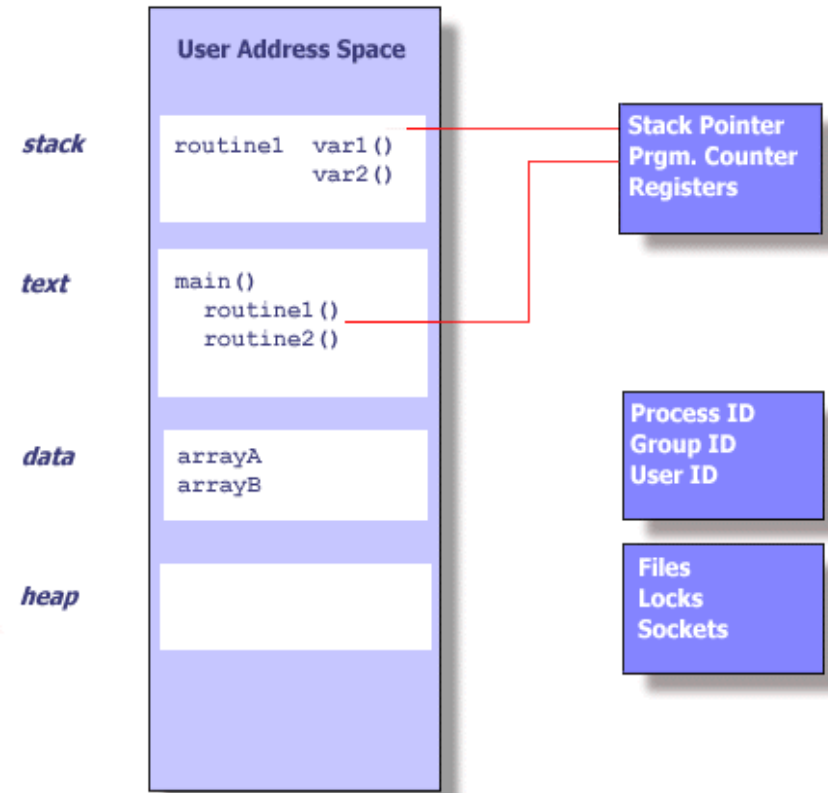
# Threads

- Processes have the following components:
  - *an address space*
  - *a collection of operating system state*
  - *a CPU context ... or thread of control*
- To use multiple CPUs on a multiprocessor system, a process would need several CPU contexts
  - *Thread fork creates new thread not memory space*
  - *Multiple threads of control could run in the same memory space on a single CPU system too!*

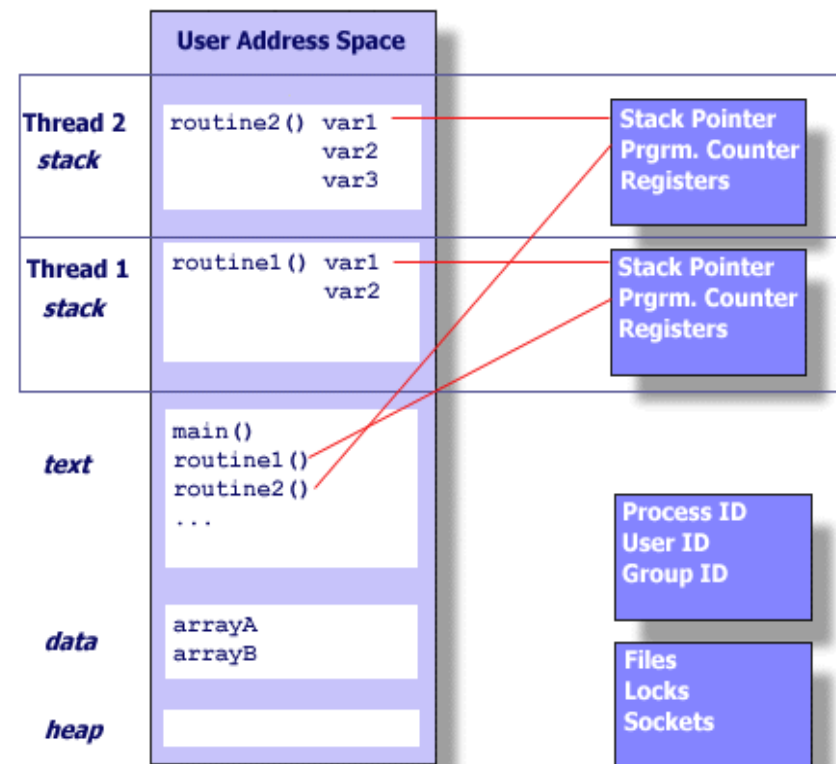
# Threads

- Threads share a process address space with zero or more other threads
- Threads have their own CPU context
  - *PC, SP, register state*
  - *Stack*
- A traditional process could be viewed as a memory address space with a single thread

# Single Thread in Address Space



# Multiple Threads in Address Space



# What Is a Thread?

- A thread executes a stream of instructions
  - *it is an abstraction for control-flow*
- Practically, it is a processor context and stack
  - *Allocated a CPU by a scheduler*
  - *Executes in a memory address space*

# Private Per-Thread State

Things that define the state of a particular flow of control in an executing program

- *Stack (local variables)*
- *Stack pointer*
- *Registers*
- *Scheduling properties (i.e., priority)*

# Shared State Among Threads

Things that relate to an instance of an executing program

- *User ID, group ID, process ID*
- *Address space:*
  - Text
  - Data (off-stack global variables)
  - Heap (dynamic data)
- *Open files, sockets, locks*



# Concurrent Access to Shared State

Changes made to shared state by one thread will be visible to the others!

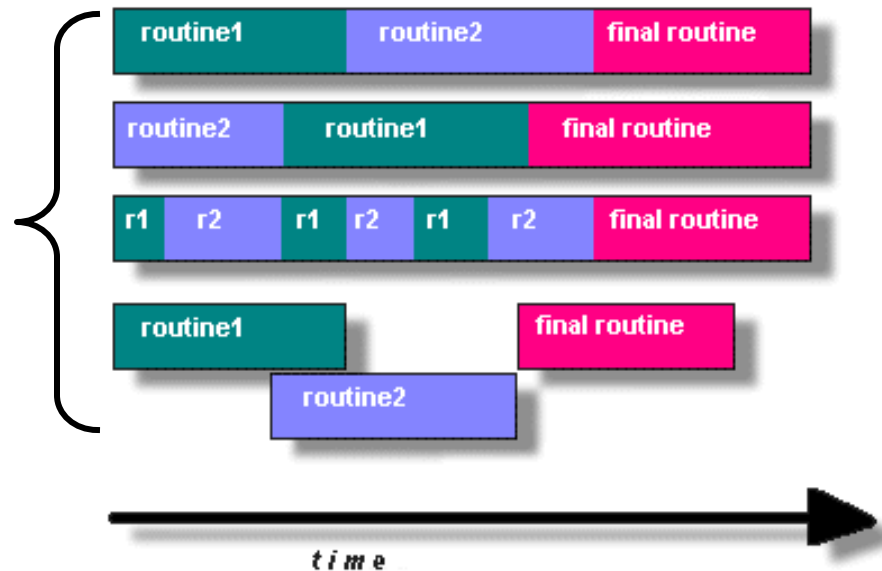
- Reading and writing memory locations requires synchronization!
- This is a major topic for later ...

# Programming With Threads

Split program into routines to execute in parallel

– *True or pseudo (interleaved) parallelism*

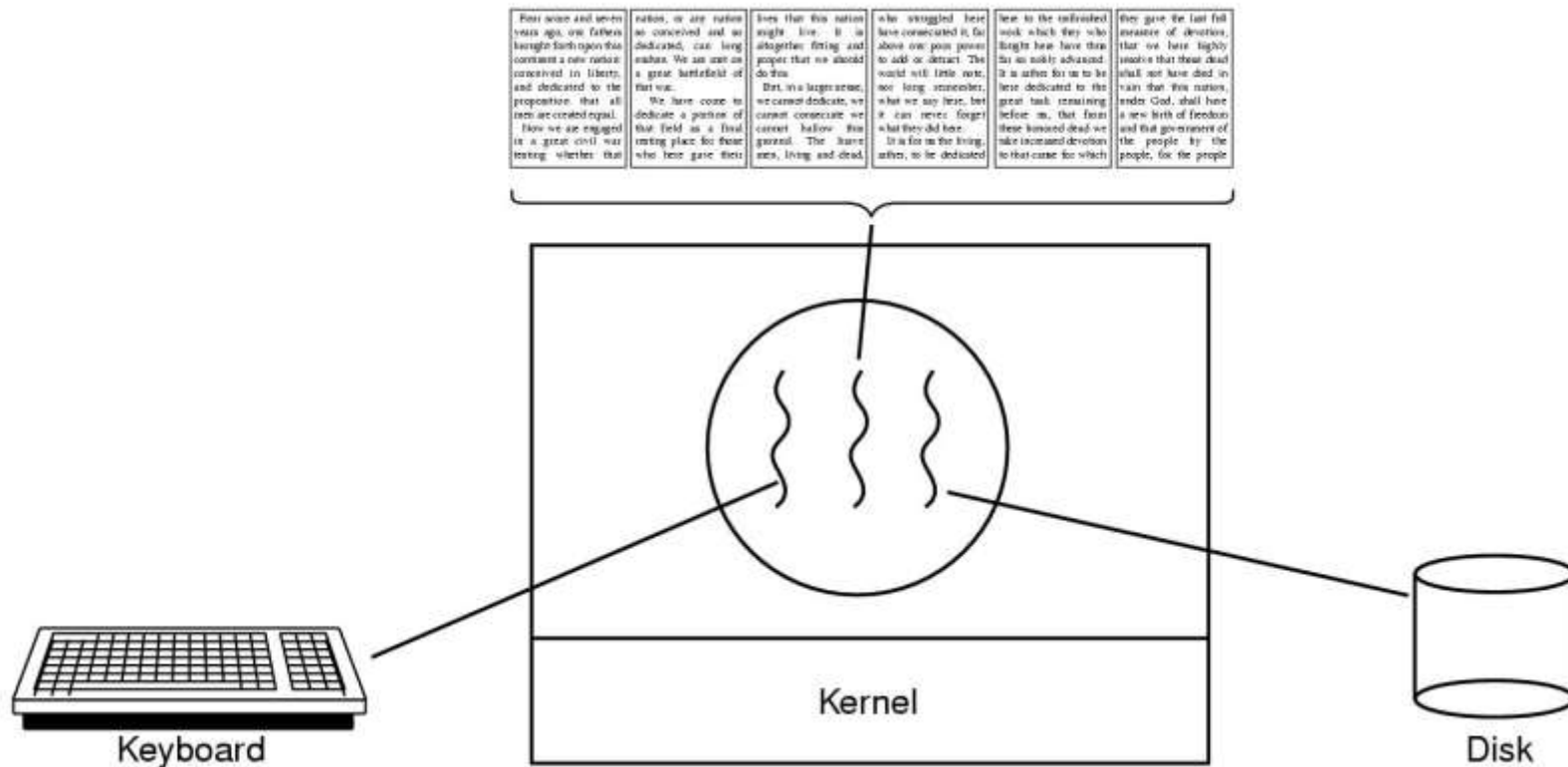
Alternative  
strategies for  
executing multiple  
routines



# Why Use Threads?

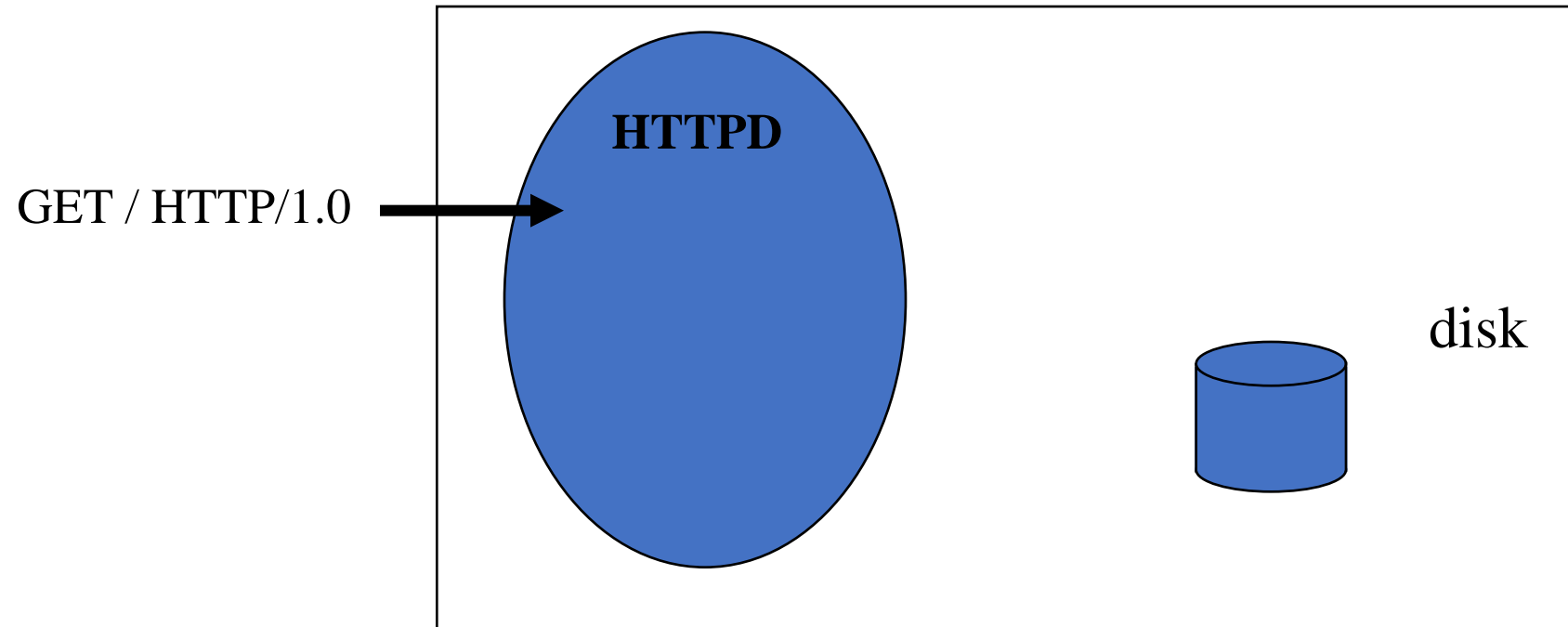
- **Utilize** multiple CPU's concurrently
- **Low cost** communication via shared memory
- Overlap computation and blocking on a single CPU
  - *Blocking due to I/O*
  - *Computation and communication*
- Handle asynchronous events

# Typical Thread Usage

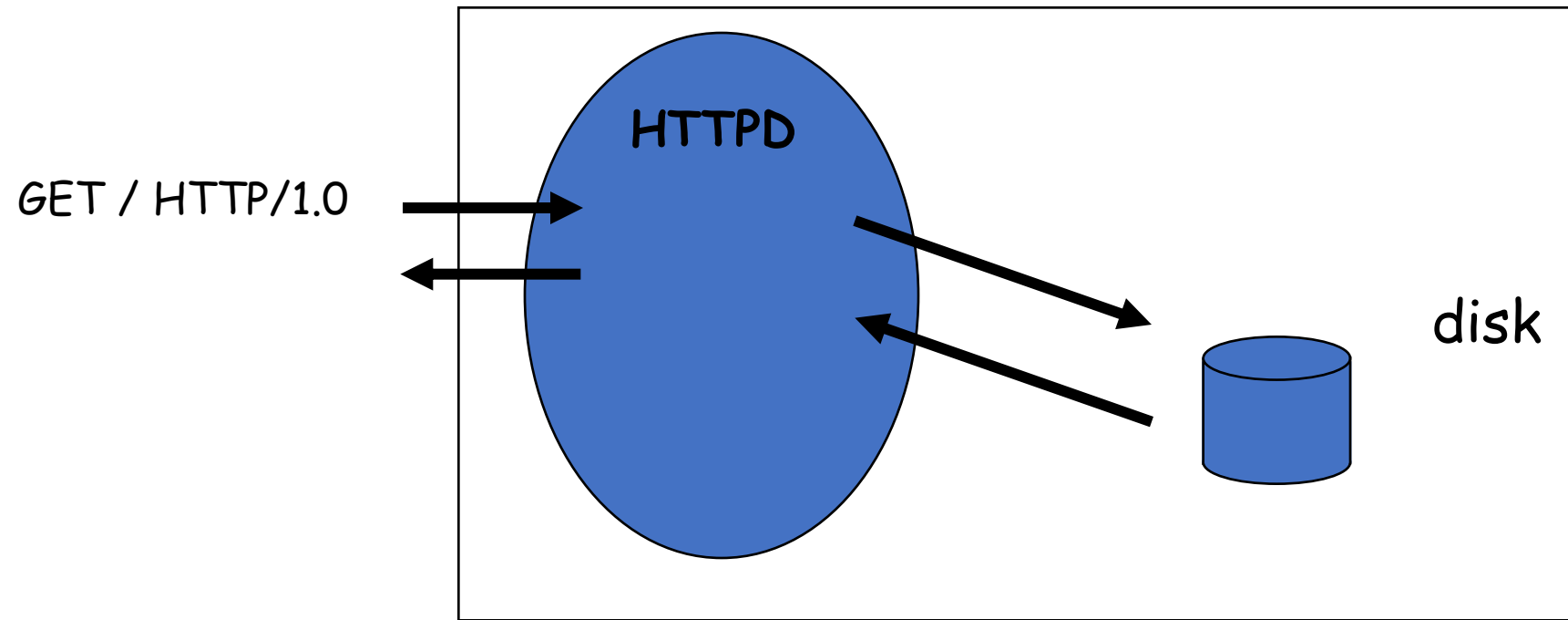


A word processor with three threads

# Processes vs Threads

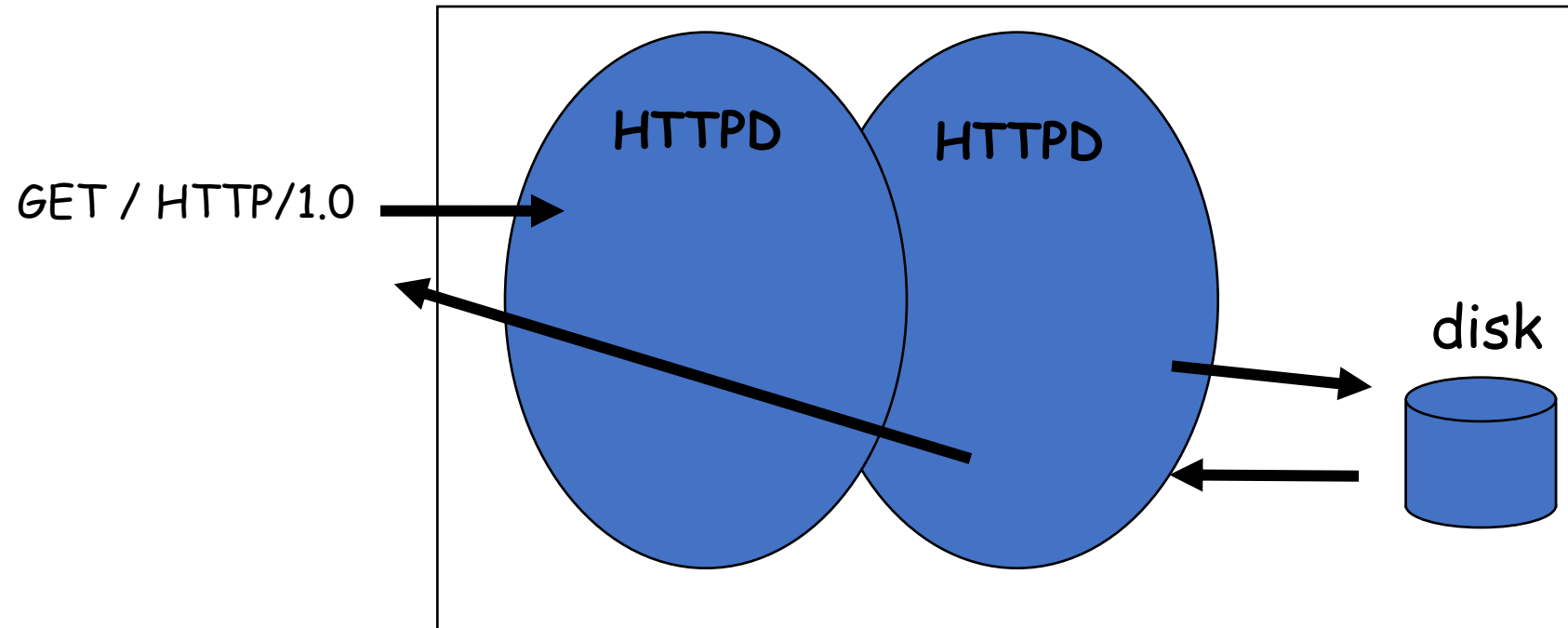


# Processes vs Threads

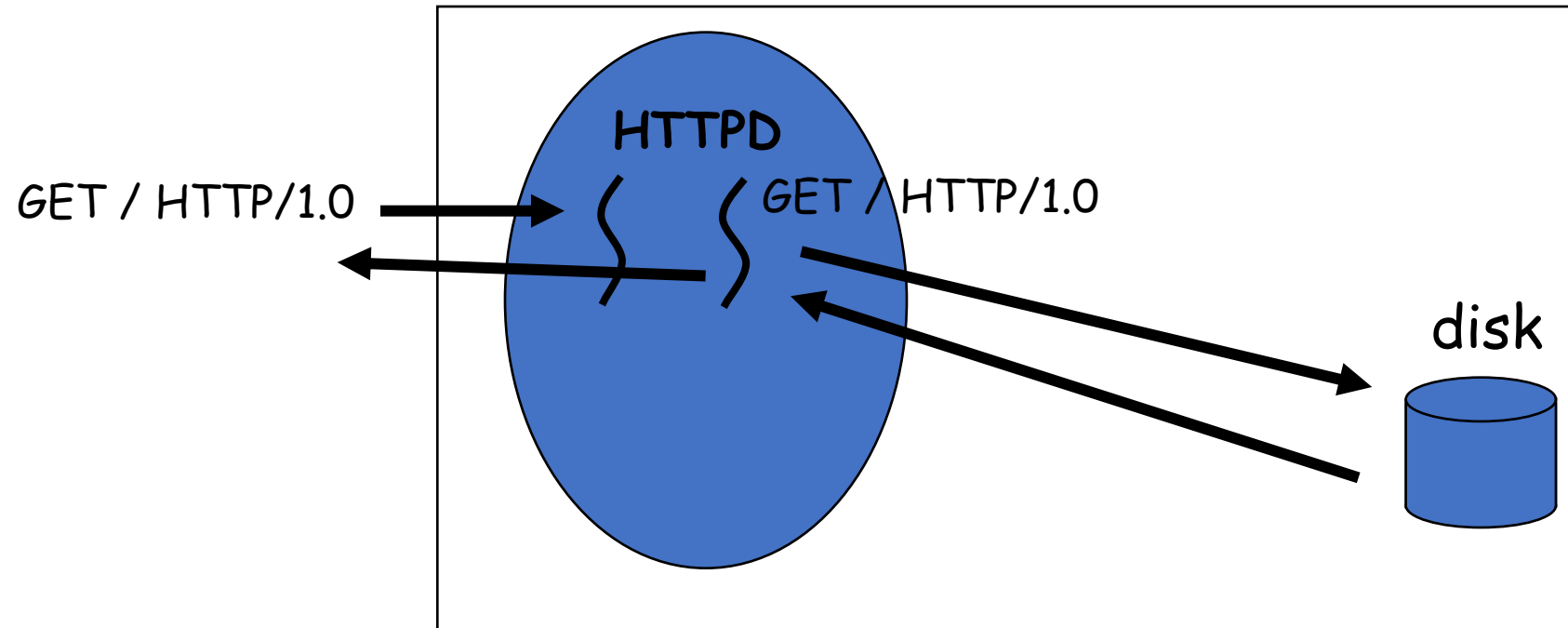


Why is this not a good web server design?

# Processes vs Threads



# Processes vs Threads





# Processes vs Threads

