EPIC
Institute of Technology
Powered by epam

Scheduling

Course: Systems Architecture

Lecturer: Gleb Lobanov

May, 2024

EPIC
Institute of Technology
Powered by epam

Contents

EPIC
Institute of Technology
Powered by epam

01

Processes

EPIC
Institute of Technology
Powered by <epam>

Processes

```
└─ ps aux
USER               PID  %CPU %MEM      VSZ    RSS   TT  STAT STARTED      TIME COMMAND
glebodin           673  11,2  1,3 410311408 223424   ??  S    чт06    4:48.31 /Applicatio
_windowserver      362   7,4  0,7 411053248 123856   ??  Ss   чт06  160:06.67 /System/Lib
glebodin           739   4,6  0,4 442387360  71472   ??  S    чт06    8:43.04 /Applicatio
glebodin          4935   2,3  1,3 1598023296 226272   ??  S    11:30    36:54.08 /Applica
d
glebodin          1904   2,0  0,0 408304000   7600 s000  S     7:27     0:10.48 -zsh
glebodin           663   1,6  2,8 443056784 471312   ??  S    чт06   21:50.40 /Applicatio
glebodin         16795   1,5  0,3 408599296  46256   ??  S     7:05     0:00.15 /System/L
glebodin          2888   1,3  1,9 10423780 310688   ??  S     8:40    37:14.66 /Users/gle
glebodin           850   1,0  0,6 10731028  93072   ??  S    чт06    5:51.41 /opt/homebre
glebodin          2899   0,9  0,9 10209980 157136   ??  Ss    8:40    11:42.95 /Users/gle
glebodin         12056   0,9  1,7 410224016 289344   ??  S     2:02    57:40.31 /Applicat
root               323   0,5  0,1 409186000  10592   ??  Ss   чт06    1:43.82 /System/Lib
glebodin         16673   0,5  0,1 409183104  23008   ??  S     7:01     0:00.55 /System/L
root               449   0,5  0,1 408996432  22224   ??  Ss   чт06    5:43.93 /usr/libexe
glebodin           670   0,5  2,7 417848288 458752   ??  S    чт06   66:15.91 /Applicatio
glebodin         16794   0,4  0,2 408606272  37696   ??  S     7:05     0:00.14 /System/L
root               308   0,3  0,2 409905424  28912   ??  Ss   чт06    4:39.81 /System/Lib
glebodin          2922   0,2  2,5 14386044 426816   ??  S     8:40     2:17.90 /Users/gle
_networkd          436   0,2  0,1 408635696  14400   ??  Ss   чт06    0:18.64 /usr/libexe
root               532   0,2  0,5 414207504  86160   ??  Ss   чт06    3:35.34 /System/Lib
glebodin         16752   0,2  0,0 408067040   4720   ??  S     7:04     0:00.22 /System/L
root               281   0,1  0,0 408632224   6752   ??  Ss   чт06    1:31.28 /System/Lib
root               510   0,1  0,3 409251792  51808   ??  SN   чт06   22:11.07 /opt/osquer
root               292   0,1  0,2 408941920  25840   ??  Ss   чт06    5:38.02 /Library/Ap
glebodin         16698   0,1  0,1 409001184  19632   ??  Ss    7:02     0:00.19 /System/L
glebodin          4926   0,1  0,5 1585149744  89792   ??  S    11:30     1:04.29 /Applica
glebodin          5543   0,1  1,1 441033632 180832   ??  S    11:53     9:37.72 /Applicat
_locationd       16729   0,0  0,0 408361504   5152   ??  S     7:04     0:00.06 /usr/libe
root             16728   0,0  0,0 408361488   5024   ??  Ss    7:04     0:00.05 /usr/libe
glebodin         16727   0,0  0,1 408630528  11296   ??  S     7:04     0:00.15 /usr/libe
```

EPIC
Institute of Technology
Powered by epam

Let’s touch them

```python
import itertools
import psutil

# getting all processes
all_processes = psutil.process_iter()

# getting 5 processes
limited_processes = itertools.islice(all_processes, 5)

# output info about processes
for process in limited_processes:
    pid = process.pid
    name = process.name()
    print(f"PID: {pid}, Name: {name}")

process = psutil.Process(16141)
print("process id:", process.pid)
print("process name:", process.name())
print("cp usage:", process.cpu_percent())
print("usage of memory:", process.memory_info().rss)
print("status:", process.status())
print("time of creation:", process.create_time())
print("parent id:", process.ppid())
```

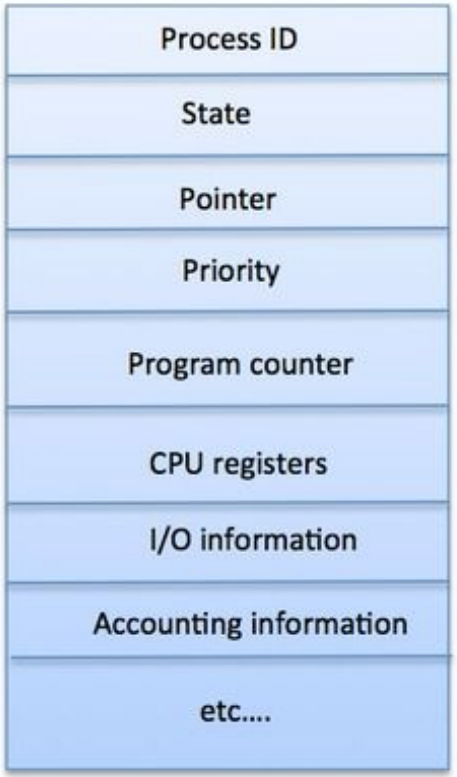
EPIC
Institute of Technology
Powered by epam

Let’s touch them

```
└─ python3 processes.py
PID: 0, Name: kernel_task
PID: 1, Name: launchd
PID: 277, Name: syslogd
PID: 278, Name: UserEventAgent
PID: 280, Name: uninstalld
process id: 16141
process name: deleted
cp usage: 0.0
usage of memory: 8601600
status: running
time of creation: 1685115830.214878
parent id: 1
```

EPIC
Institute of Technology
Powered by epam

Process info



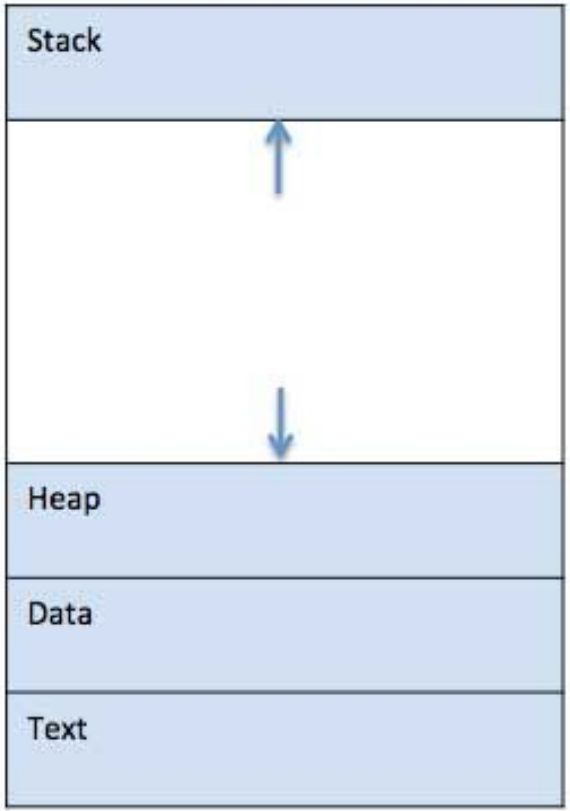
EPIC
Institute of Technology
Powered by <epam>

Priority - NI. The smaller the "nice" value, the higher the priority of the process.

```
     F S UID  PID PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
241801 S 200 7032 7286   0  60 20 1b4c   108          pts/2  0:00 ksh
200801 S 200 7568 7032   0  70 25 2310    88  5910a58  pts/2  0:00 vmstat
241801 S 200 8544 6494   0  60 20 154b   108          pts/0  0:00 ksh
```

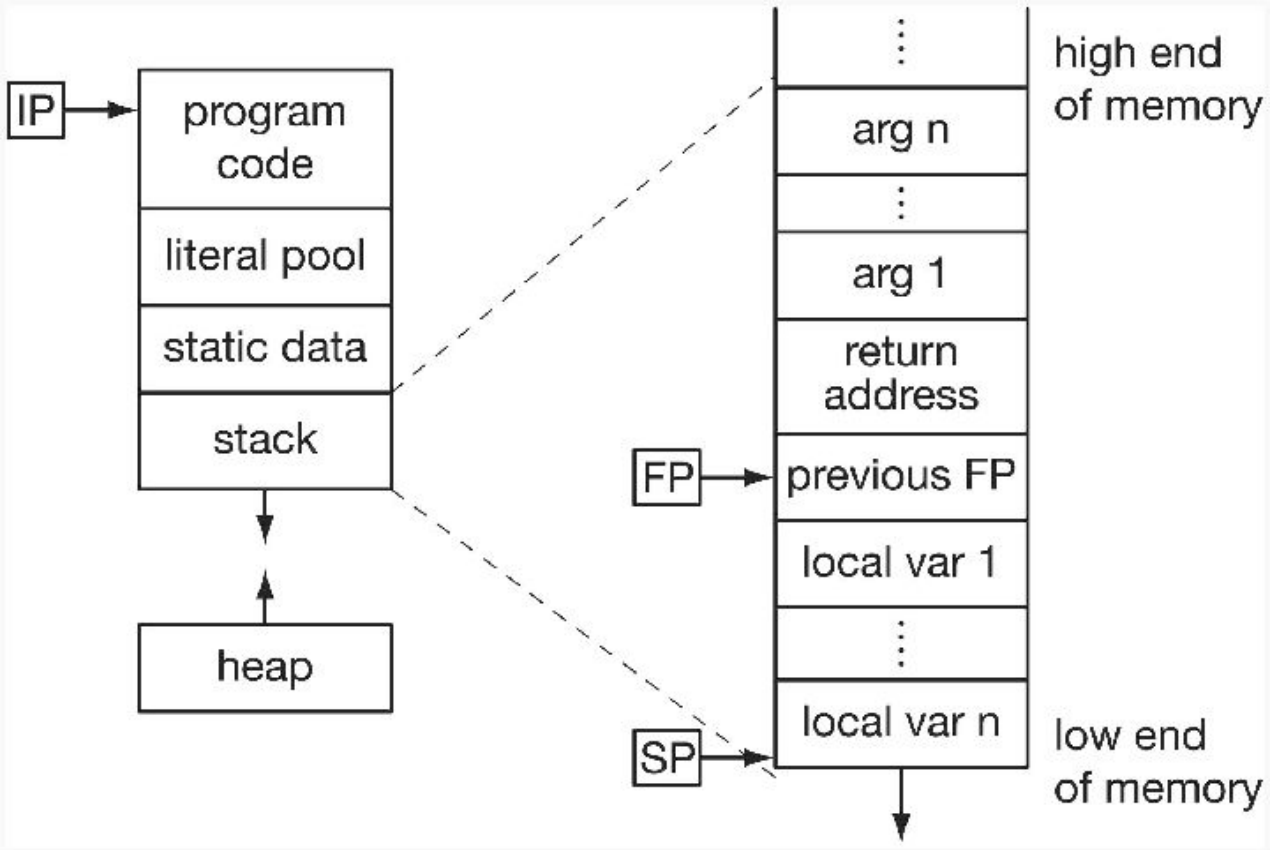
EPIC
Institute of Technology
Powered by epam

Process stack



EPIC
Institute of Technology
Powered by epam

Context



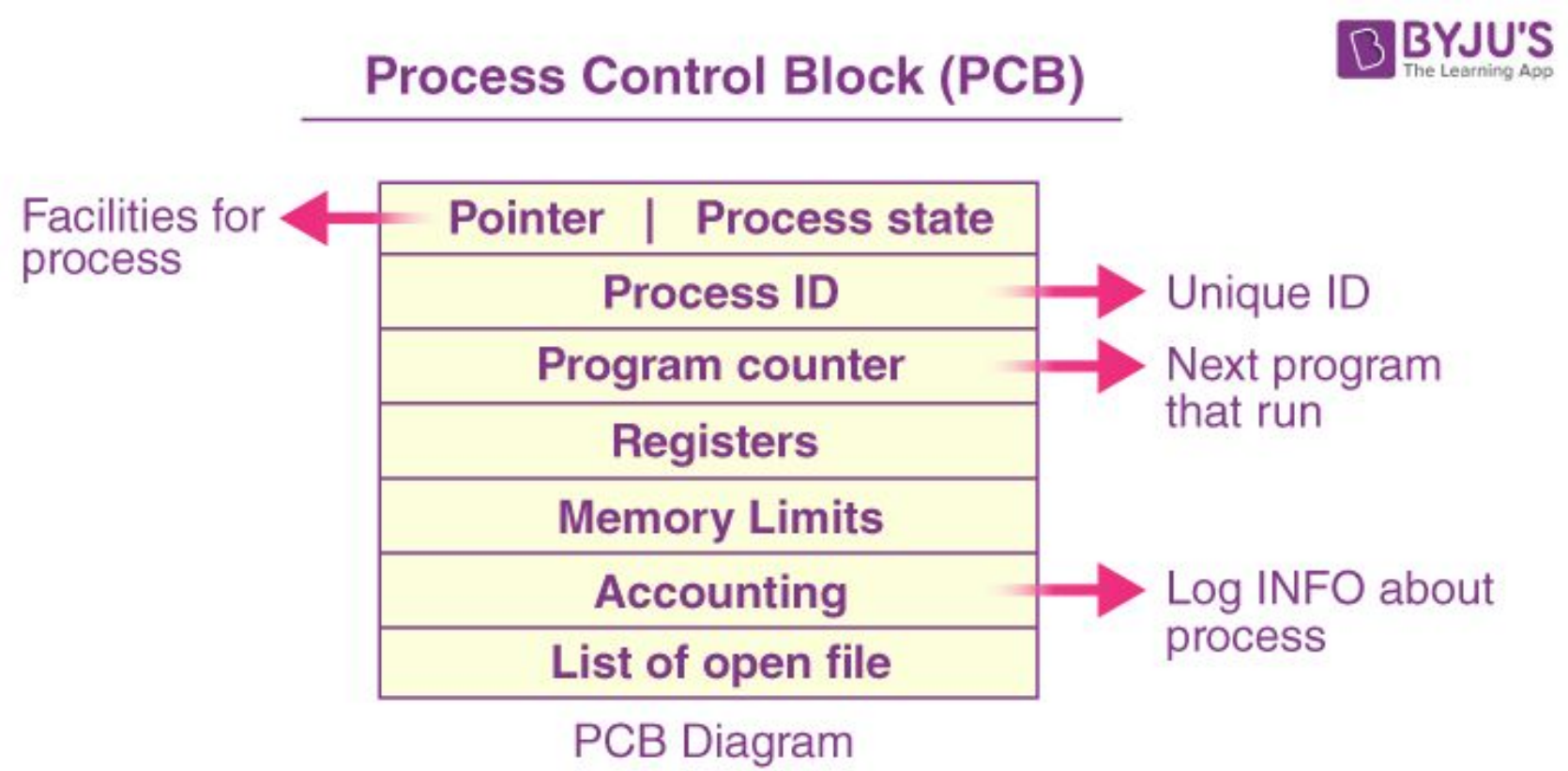
EPIC Institute of Technology
Powered by epam

Component context

1) General Purpose Registers
2) Command Counter
3) Stack context

EPIC
Institute of Technology
Powered by epam

Process



EPIC
Institute of Technology
Powered by epam

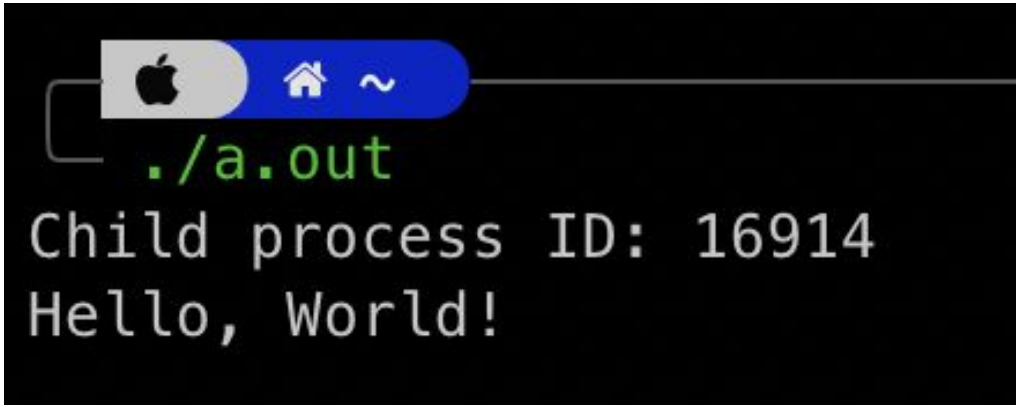
C++

not perfect)

```cpp
#include <iostream>
#include <cstdlib>
#include <unistd.h>

int main() {
    // Creating a new process
    pid_t pid = fork();

    if (pid == -1) {
        // Handling error in process creation
        std::cerr << "Error creating a process!" << std::endl;
        return 1;
    } else if (pid == 0) {
        // Code executed in the child process
        // Printing "Hello, World!"
        std::cout << "Hello, World!" << std::endl;
    } else {
        // Code executed in the parent process
        // Outputting child process ID
        std::cout << "Child process ID: " << pid << std::endl;
        // Waiting for the child process to finish
        int status;
        waitpid(pid, &status, 0);
    }
    return 0;
}
```

EPIC
Institute of Technology
Powered by <epam>

Result



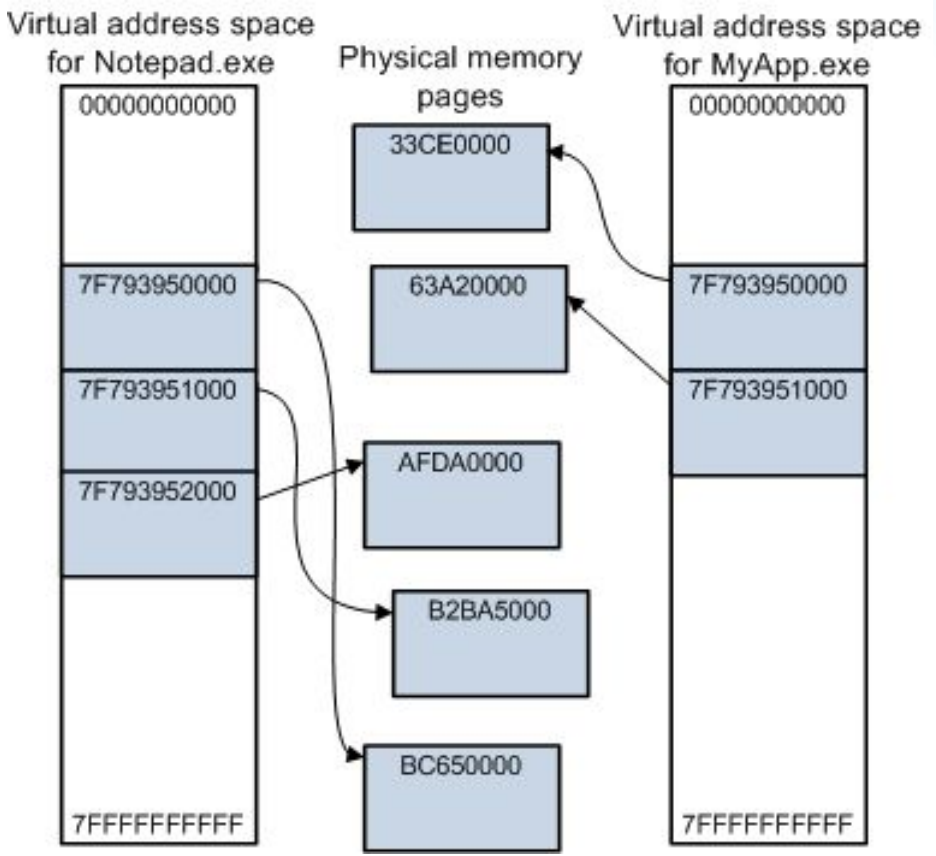
EPIC
Institute of Technology
Powered by <epam>

02

Protection

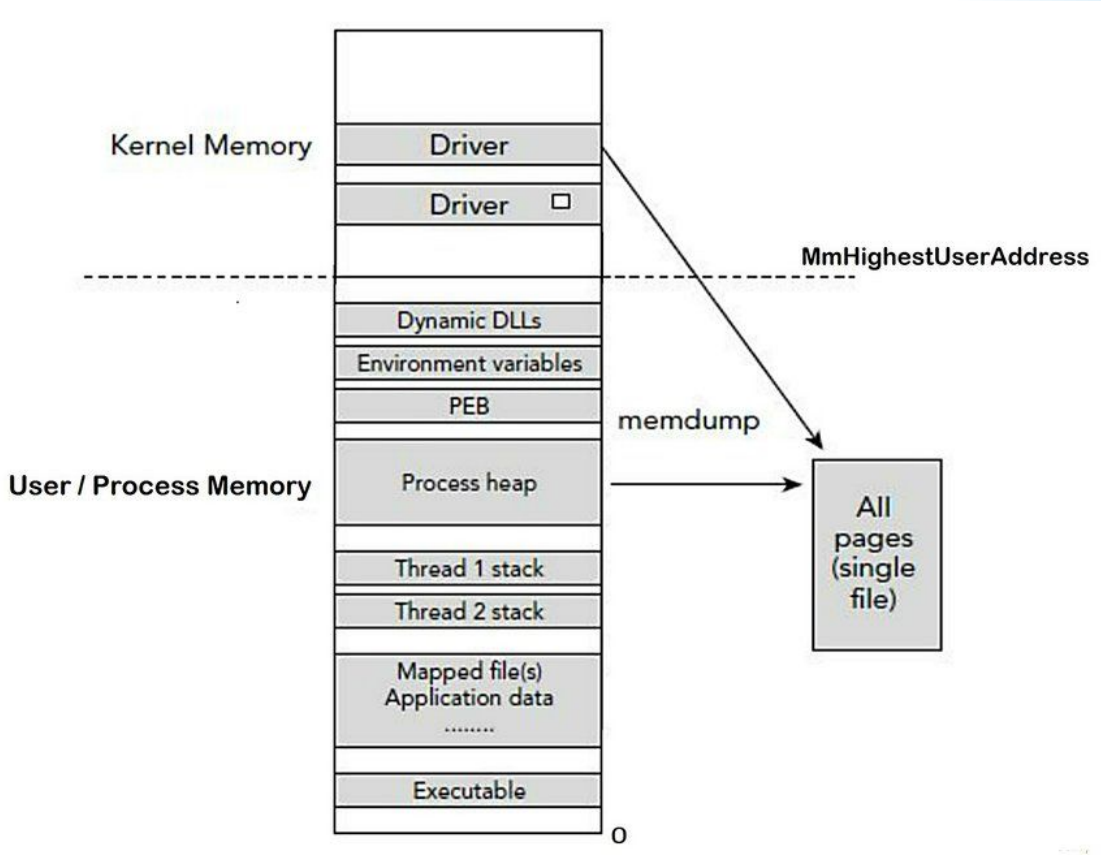
EPIC Institute of Technology
Powered by <epam>

Virtual address space



EPIC
Institute of Technology
Powered by <epam>

Virtual address space



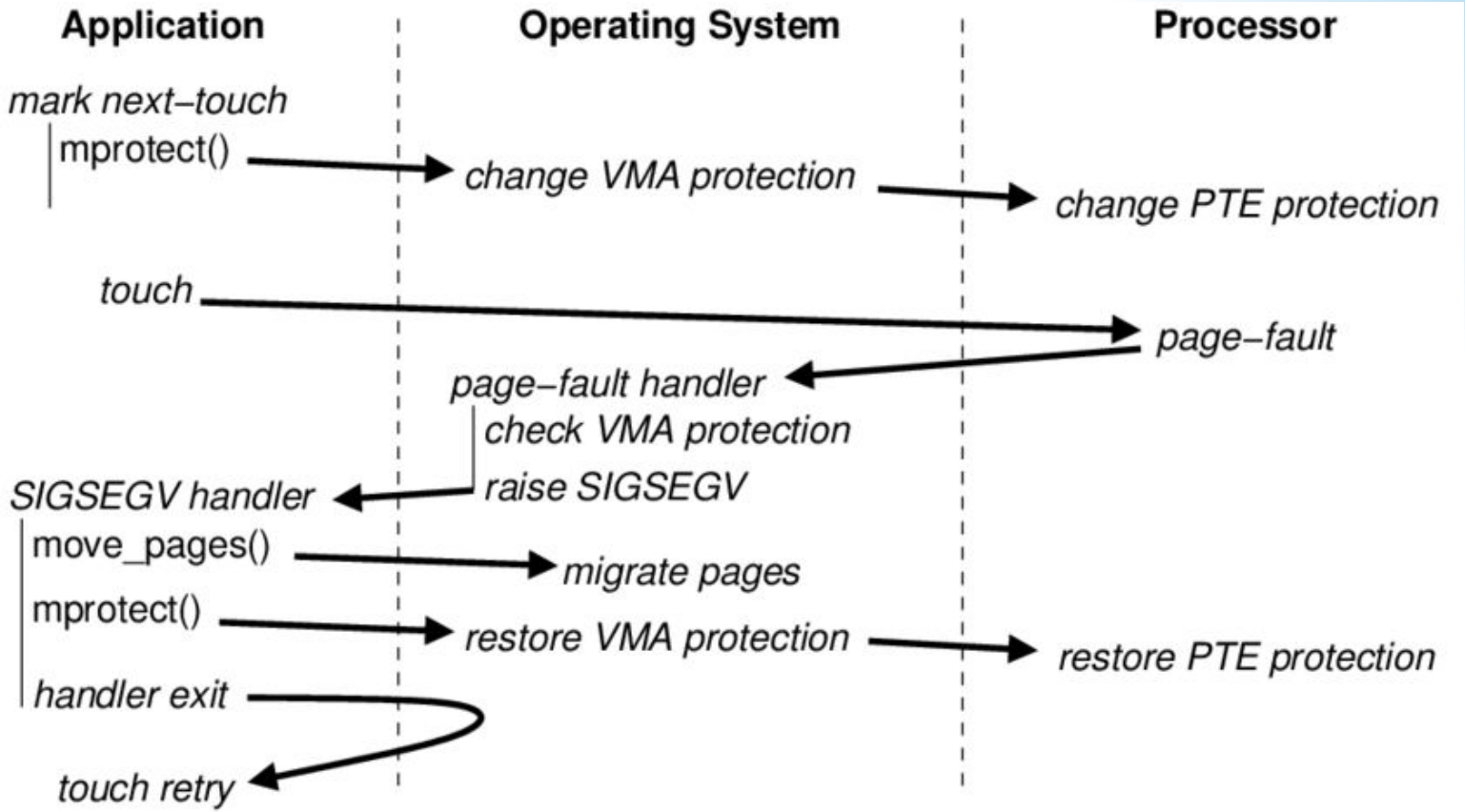
EPIC
Institute of Technology
Powered by epam

Protection

```
MPROTECT(2)                BSD System Calls Manual               MPROTECT(2)

NAME
     mprotect -- control the protection of pages

SYNOPSIS
     #include <sys/mman.h>

     int
     mprotect(void *addr, size_t len, int prot);

DESCRIPTION
     The mprotect() system call changes the specified pages to have protection prot.  Not all implementations will guarantee protection on a page basis but Mac
     OS X's current implementation does.

     When a program violates the protections of a page, it gets a SIGBUS or SIGSEGV signal.

     Currently prot can be one or more of the following:

     PROT_NONE   No permissions at all.
     PROT_READ   The pages can be read.
     PROT_WRITE  The pages can be written.
     PROT_EXEC   The pages can be executed.

RETURN VALUES
     Upon successful completion, a value of 0 is returned.  Otherwise, a value of -1 is returned and errno is set to indicate the error.

ERRORS
     mprotect() will fail if:

     [EACCES]           The requested protection conflicts with the access permissions of the process on the specified address range.

     [EINVAL]           addr is not a multiple of the page size (i.e.  addr is not page-aligned).

     [ENOMEM]           The specified address range is outside of the address range of the process or includes an unmapped page.

     [ENOTSUP]          The combination of accesses requested in prot is not supported.
```

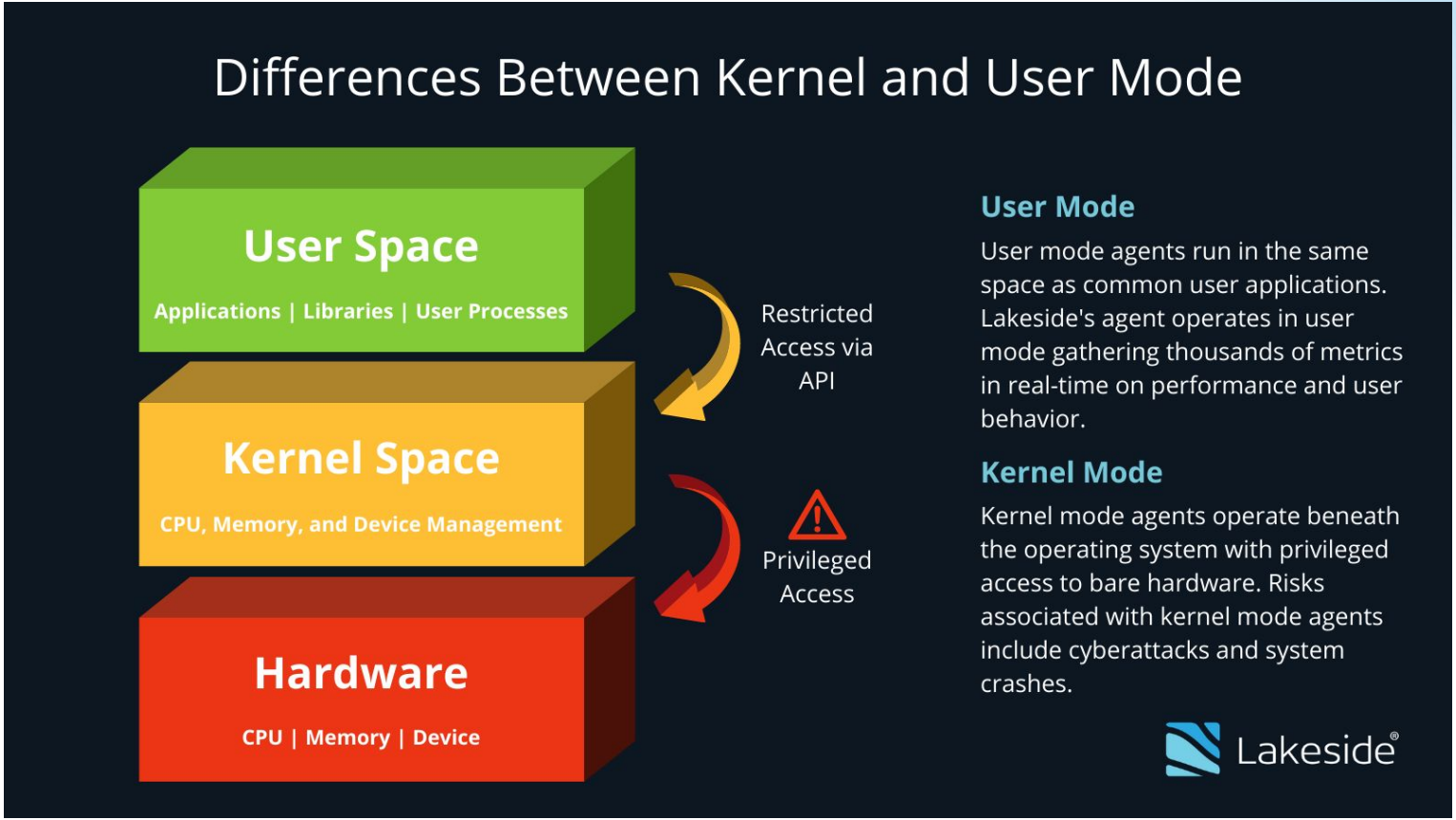
EPIC Institute of Technology
Powered by epam

Protection



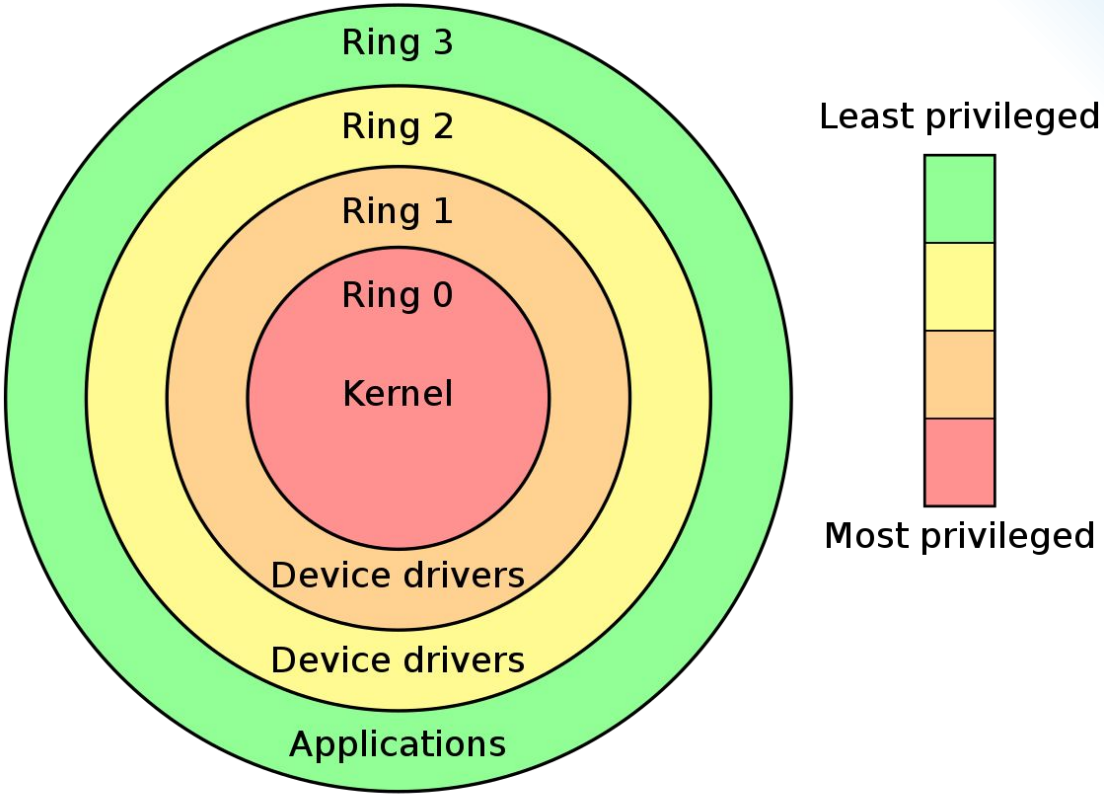
EPIC
Institute of Technology
Powered by epam

Modes



EPIC
Institute of Technology
Powered by epam

Rings



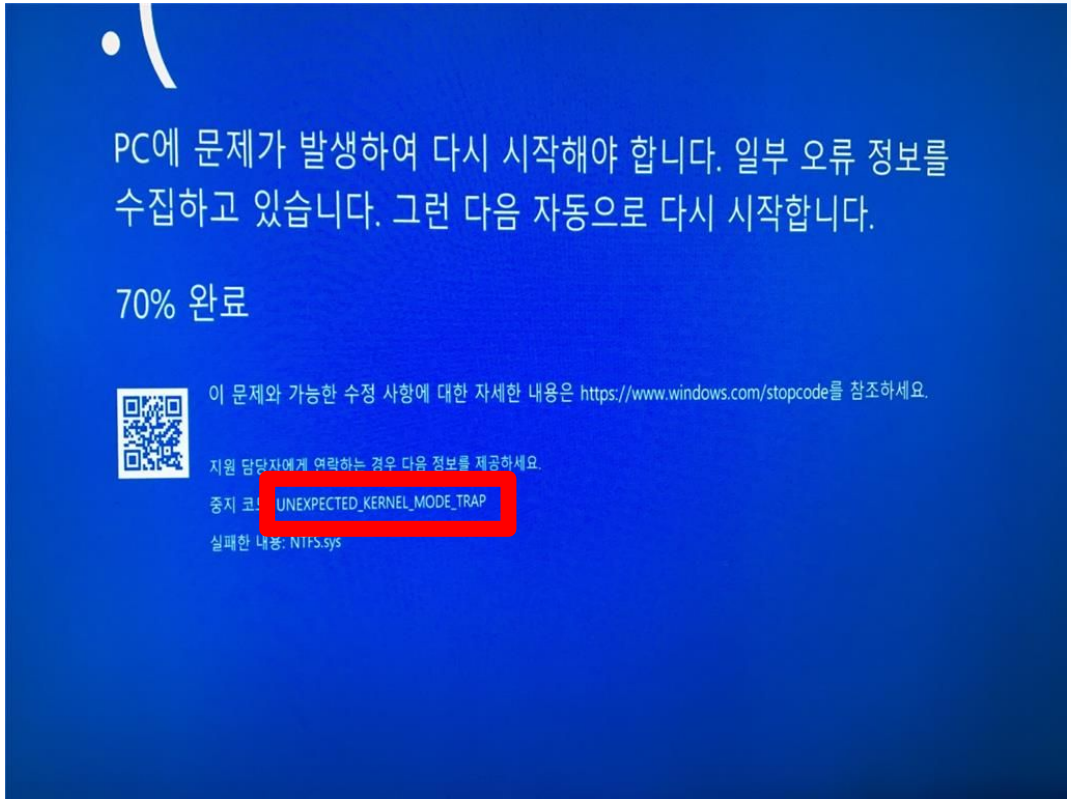
EPIC Institute of Technology
Powered by epam

Access of processes

1) Separation of execution modes
2) Permissions to access memory pages
3) Syscalls
4) Security modules and policies

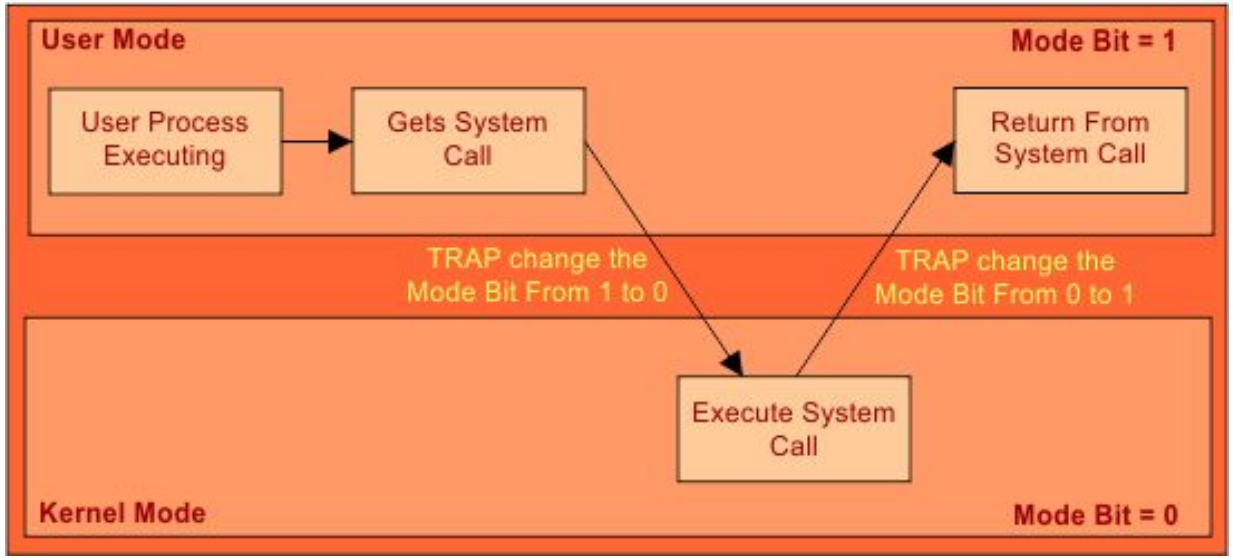
EPIC
Institute of Technology
Powered by <epam>

Blue screen



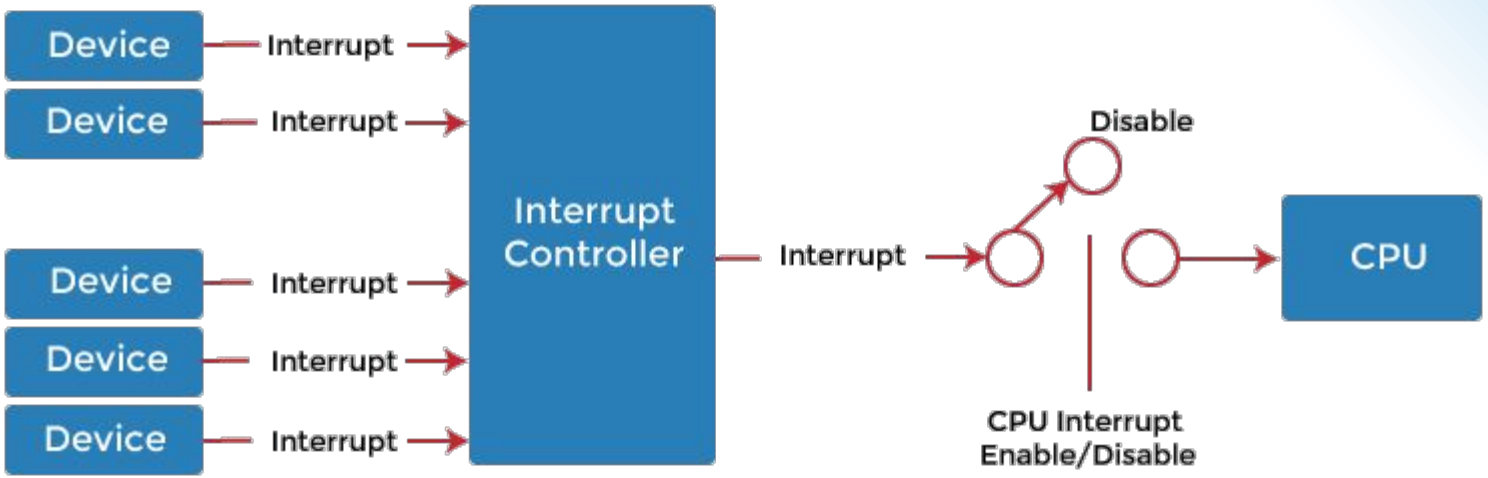
EPIC
Institute of Technology
Powered by <epam>

Syscalls



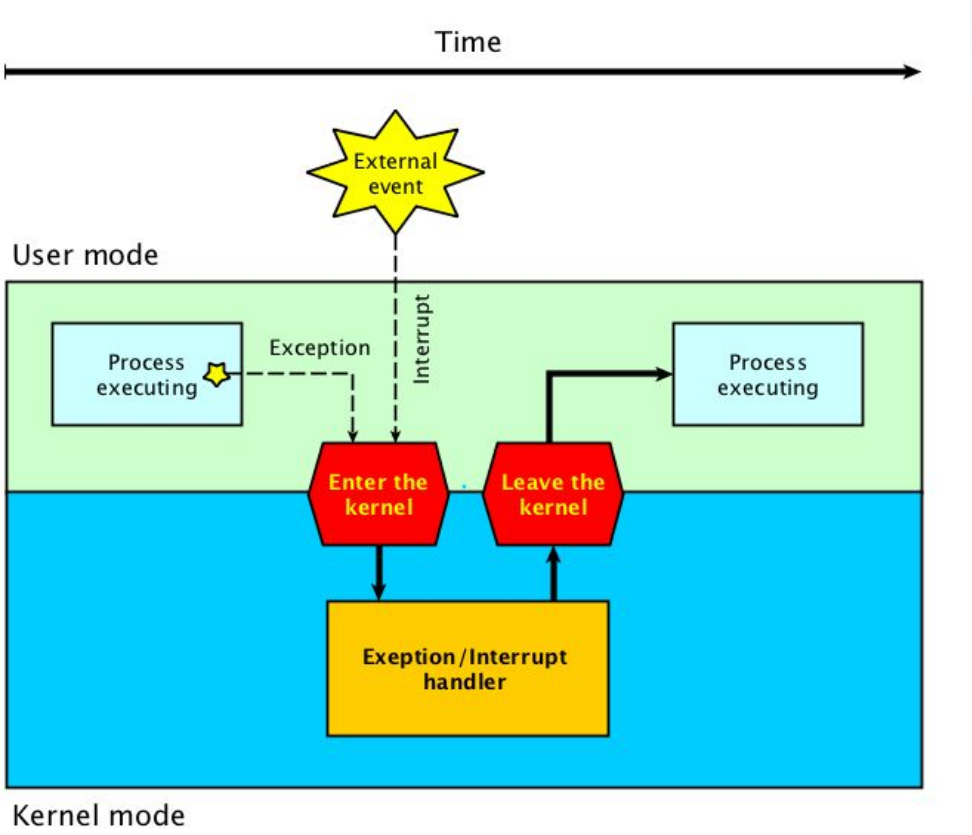
EPIC
Institute of Technology
Powered by <epam>

Interrupt



EPIC
Institute of Technology
Powered by <epam>

Interrupt handling



EPIC
Institute of Technology
Powered by epam

Interrupt types

1) Hardware Interrupts
2) Software Interrupts
3) External Interrupts
4) Timer Interrupts
5) Virtual Interrupts

EPIC Institute of Technology
Powered by epam

Timer interrupts types

1) Task Scheduling
2) System time update
3) Operations with precise time

EPIC
Institute of Technology
Powered by epam

Interrupt

```cpp
#include <iostream>

int main() {
    std::cout << "Before interrupt." << std::endl;

    // Calling a software interrupt
    asm volatile("int $3");  // Interrupt number 3

    std::cout << "After interrupt." << std::endl;

    return 0;
}
```

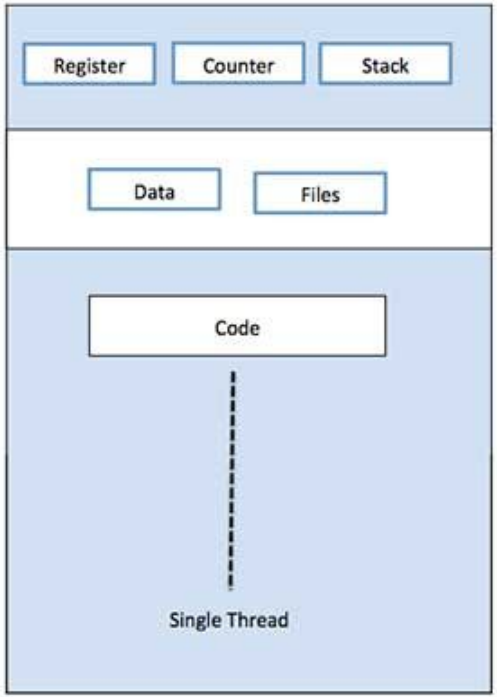
EPIC
Institute of Technology
Powered by <epam>

03

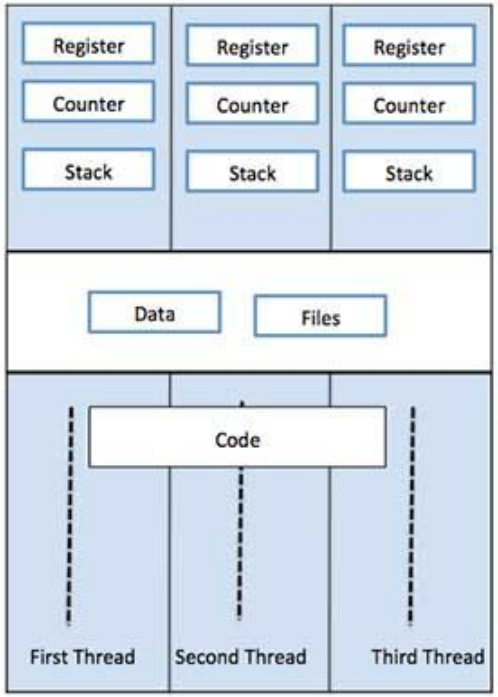
Threads

EPIC
Institute of Technology
Powered by epam

Threads



Single Process P with single thread



Single Process P with three threads

EPIC Institute of Technology
Powered by <epam>

Process vs Thread

Classification	Process	Threads
Definition	Execution of any program is a process	Segment or subset of a process is a thread
Communication	It takes more time to communicate between the different processes.	It takes less time for the communications
Resources	It consumes more resources	It consumes fewer resources
Memory	The process does not share the memory and is carried out alone	Threads are used to share their memory
Data Sharing	Process does not share their data	Threads share their data
Size	The process consumes more size	Threads are lightweight
Execution error	One process will not affect any other process due to an error	Threads will not run if one thread has an error
Termination time	Process usually takes more time to terminate	Threads take less time to terminate

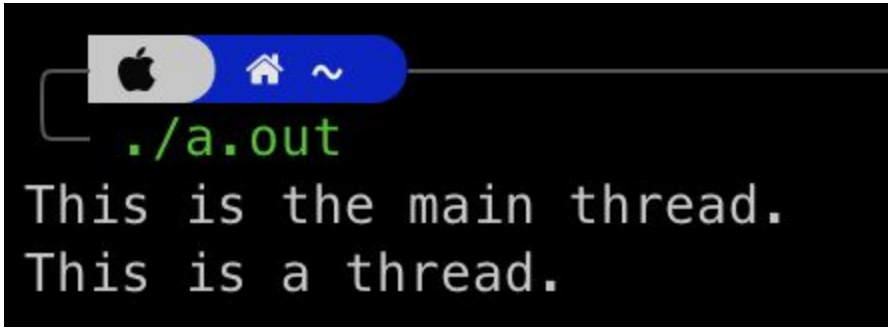
EPIC Institute of Technology
Powered by epam

Threads code

```cpp
#include <iostream>
#include <thread>

// Function to be executed by the thread
void threadFunction() {
    std::cout << "This is a thread." << std::endl;
}

int main() {
    // Creating a thread object and passing it the function to execute
    std::thread myThread(threadFunction);

    std::cout << "This is the main thread." << std::endl;

    // Waiting for the thread to finish
    myThread.join();

    return 0;
}
```

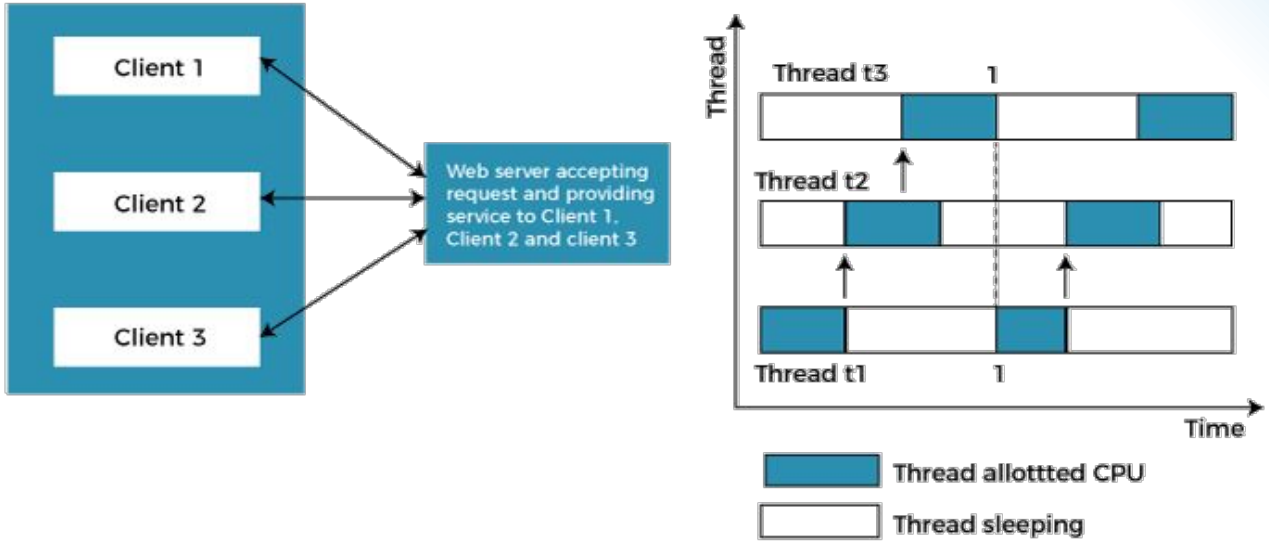
EPIC
Institute of Technology
Powered by <epam>

Result



EPIC
Institute of Technology
Powered by epam

Threads control



EPIC Institute of Technology
Powered by <epam>

Threads control in OS

1) Creating streams
2) Scheduling flows
3) Context switching
4) Synchronization and interaction
5) Priority Management
6) Ending Flows

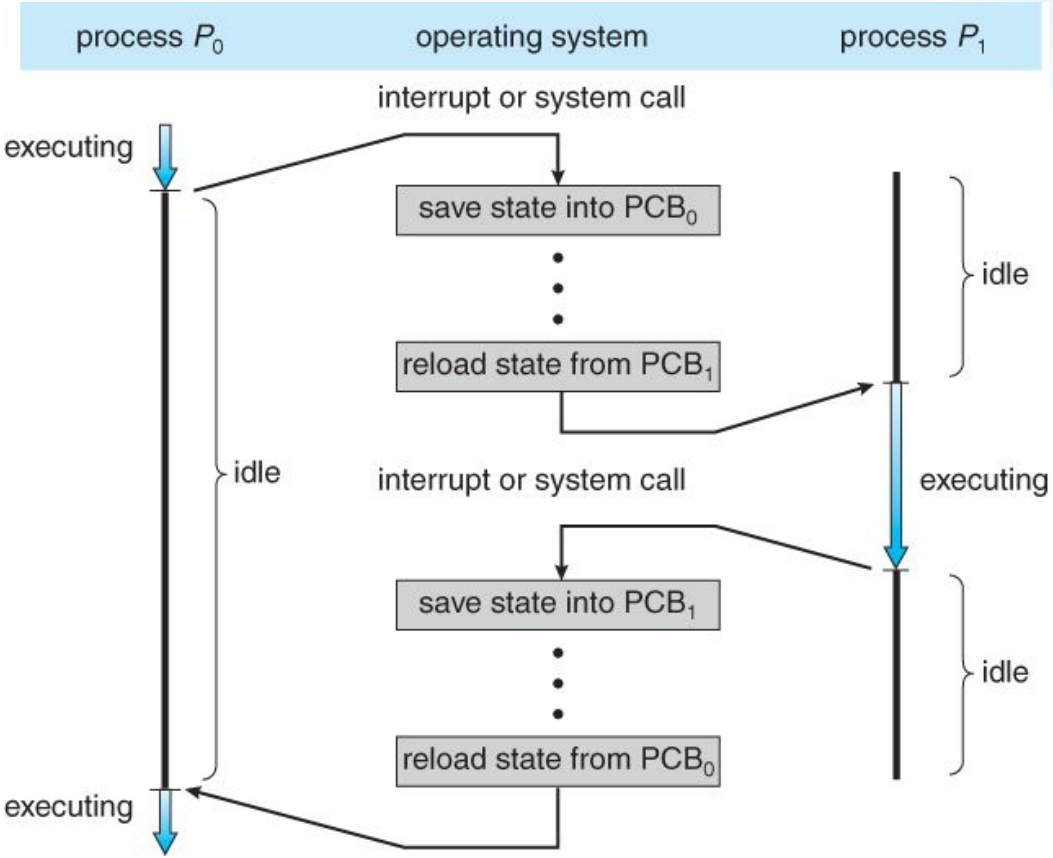
EPIC Institute of Technology
Powered by <epam>

Why we can’t make it simple?

1) Poor use of resources
2) Parallel computing
3) Responsiveness
4) Synchronization and collaboration
5) Multitasking support

EPIC
Institute of Technology
Powered by epam

Switching



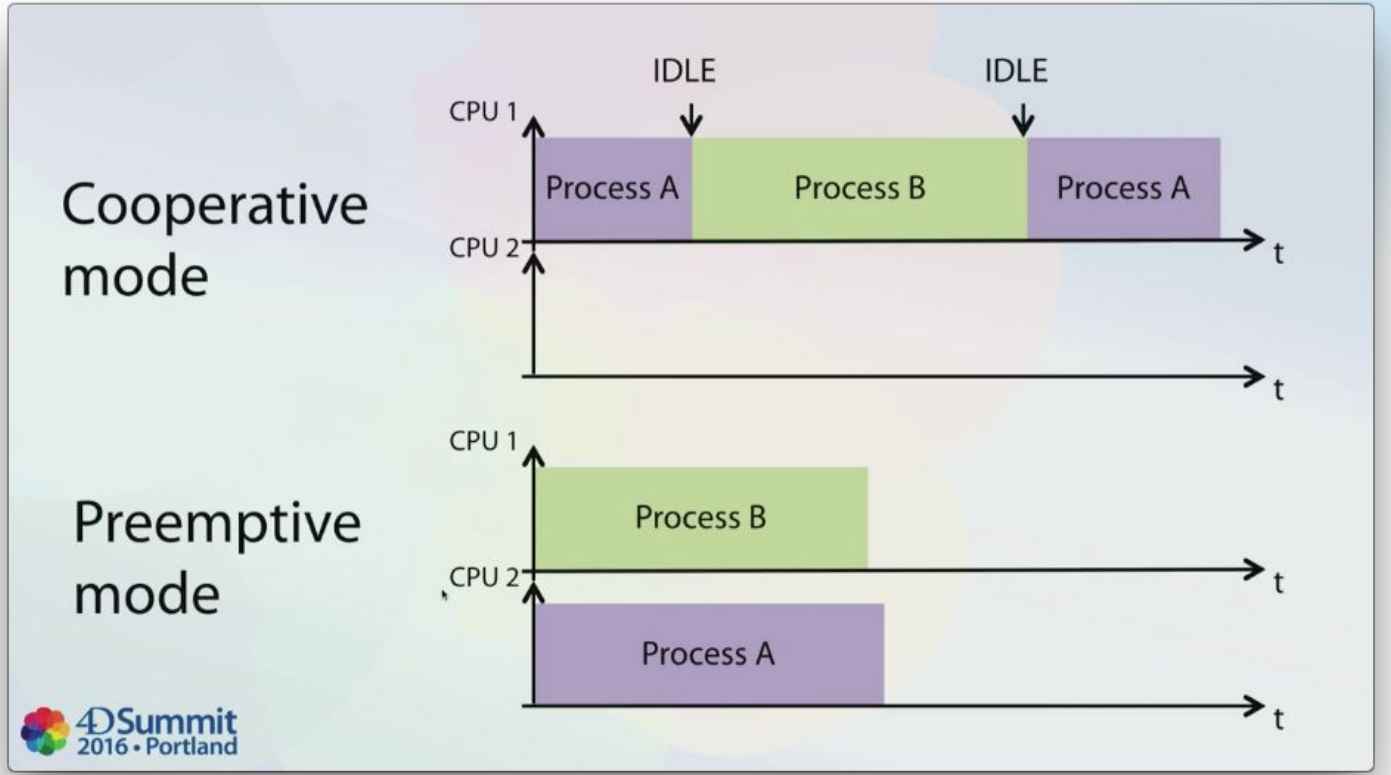
EPIC
Institute of Technology
Powered by epam

How to change task?

1) Cooperative multitasking
2) Interruptions by time interval
3) Event-based interruptions
4) I/O multiplexing

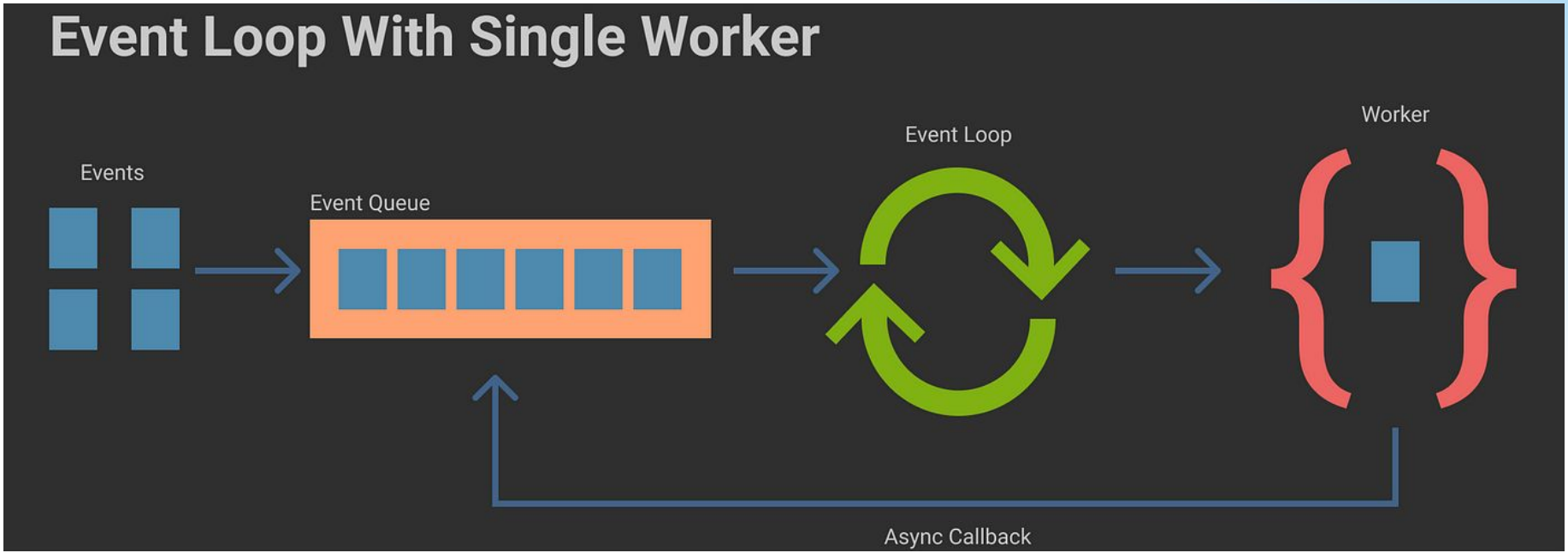
EPIC
Institute of Technology
Powered by epam

Multitasking



EPIC
Institute of Technology
Powered by <epam>

Event loop



EPIC
Institute of Technology
Powered by <epam>

04

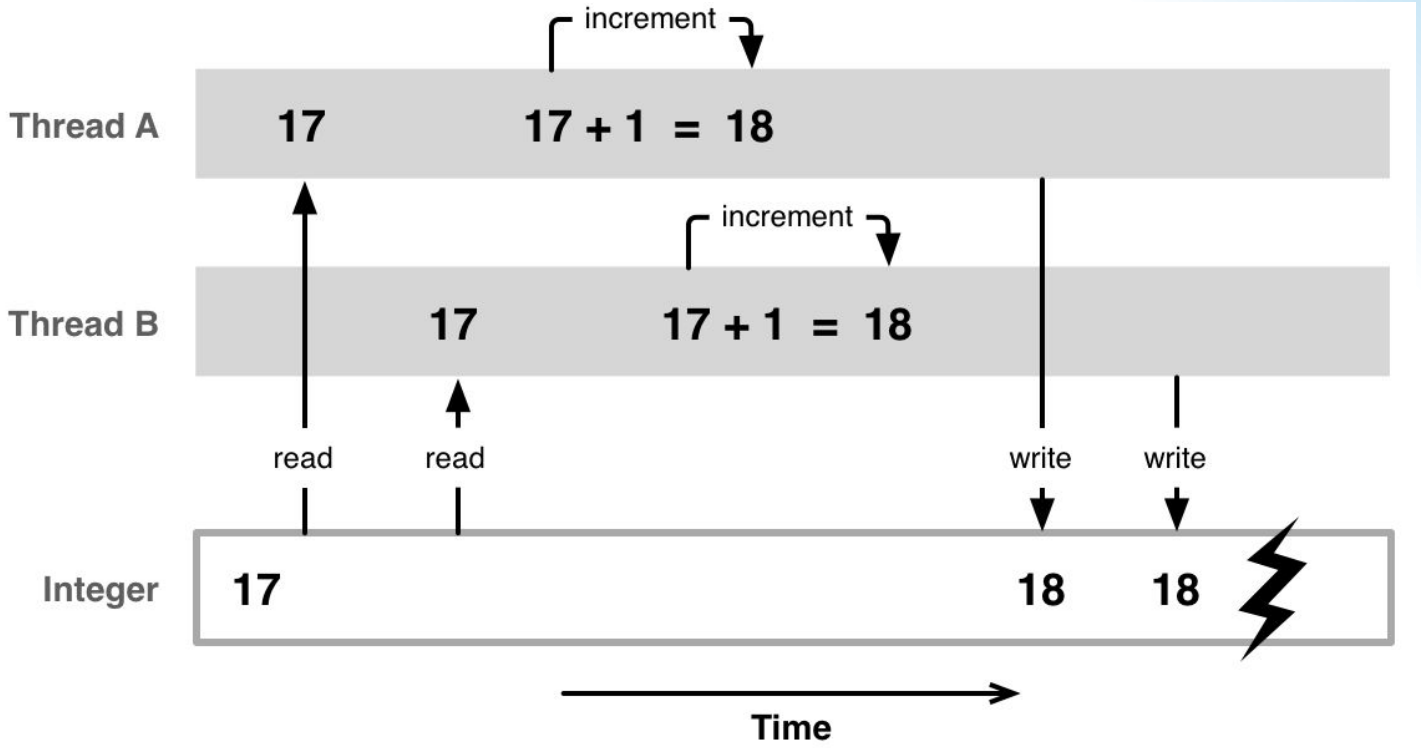
Race condition

EPIC
Institute of Technology
Powered by epam

Race condition

EPIC
Institute of Technology
Powered by epam

Race condition



EPIC Institute of Technology
Powered by epam

Bug bounty

Most of OAuth 2 API implementations seem to have multiple Race Condition vulnerabilities for processing requests for Access Token or Refresh Token.

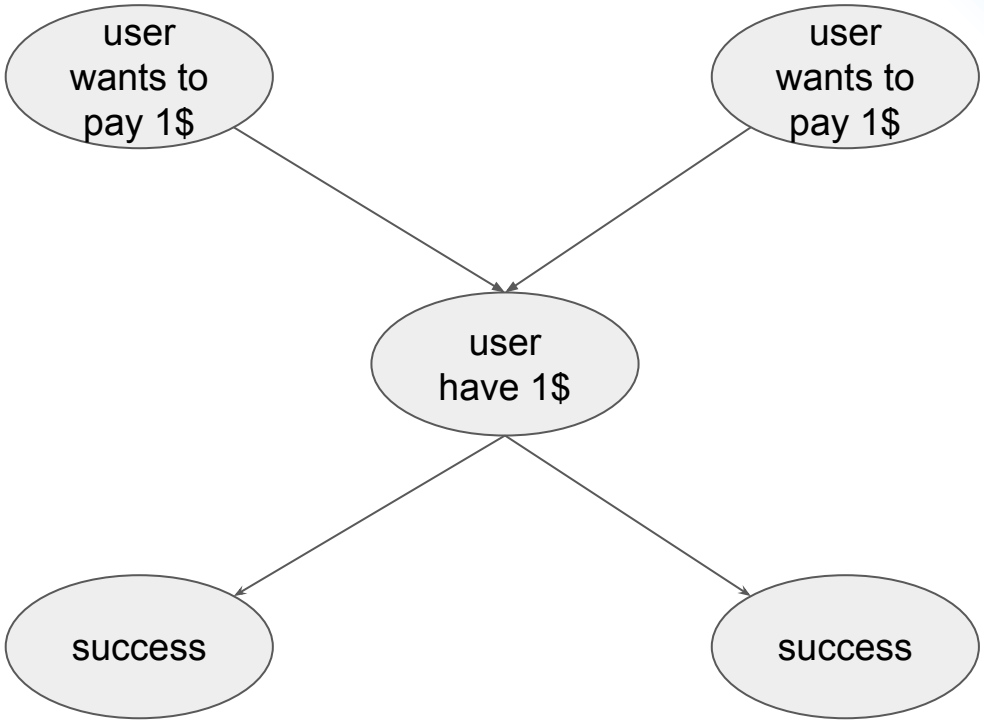
Race Condition allows a malicious application to obtain several `access_token` and `refresh_token` pairs while only one pair should be generated. Further, it leads to authorization bypass when access would be revoked.

I've already tested for this vulnerability 11 different targets (web-services providing OAuth2 API), and 6 of them are vulnerable. Only one target seems to be certainly protected (or I just failed to catch into Race Condition time window). The rest 4 targets have Race Condition bug, but protected against further exploitation by one of the following reasons:

- only one of several tokens generated is valid (**1 target**)
- for any access revocation all tokens always are revoked (**1 target**)
- for all concurring requests finished successfully the same `access_token` values (or in pair with `refresh_token`) obtained (**2 targets**)

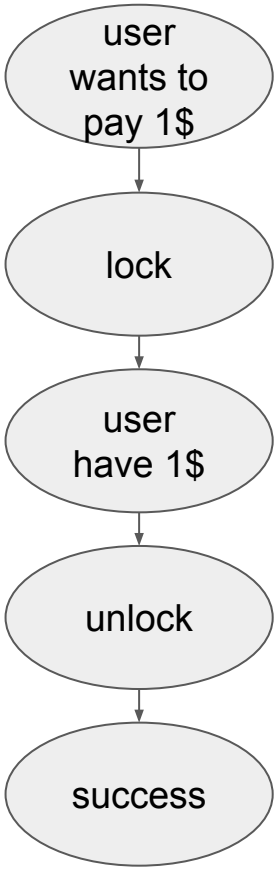
EPIC
Institute of Technology
Powered by epam

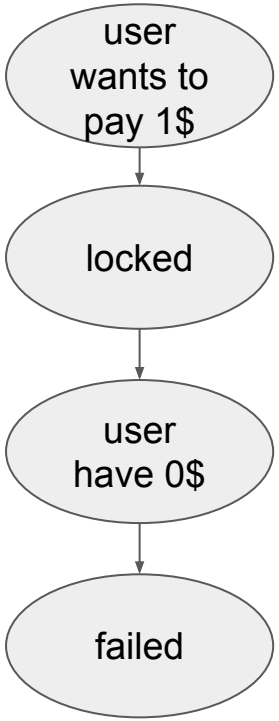
Banking



EPIC
Institute of Technology
Powered by epam

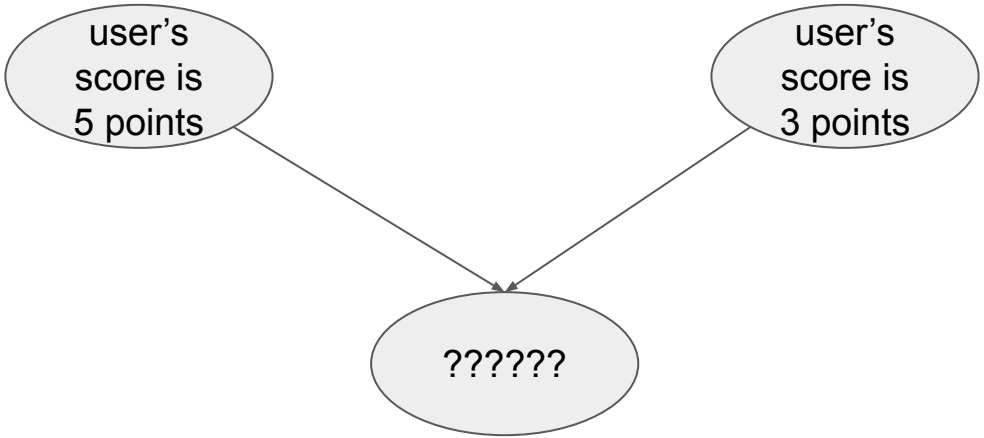
Solution





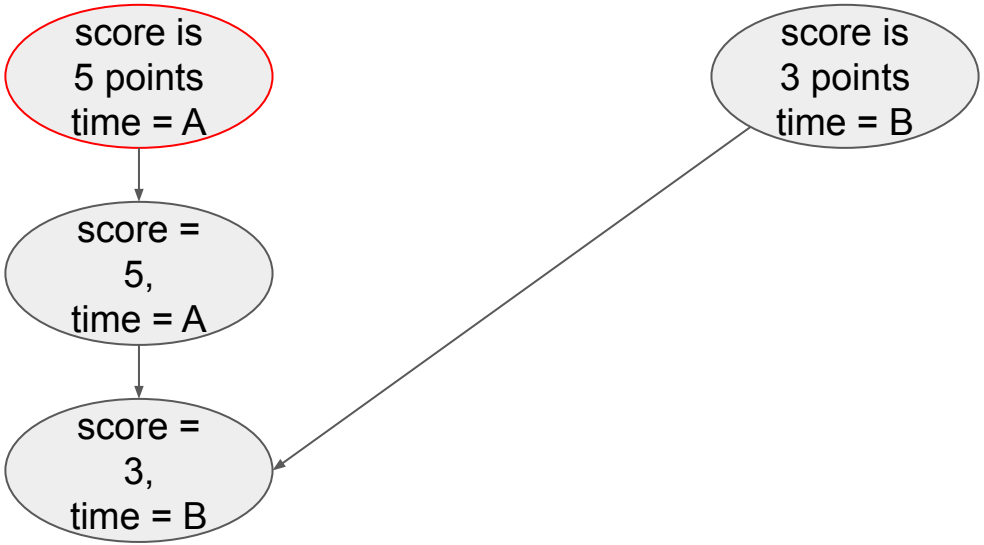
EPIC
Institute of Technology
Powered by epam

Game



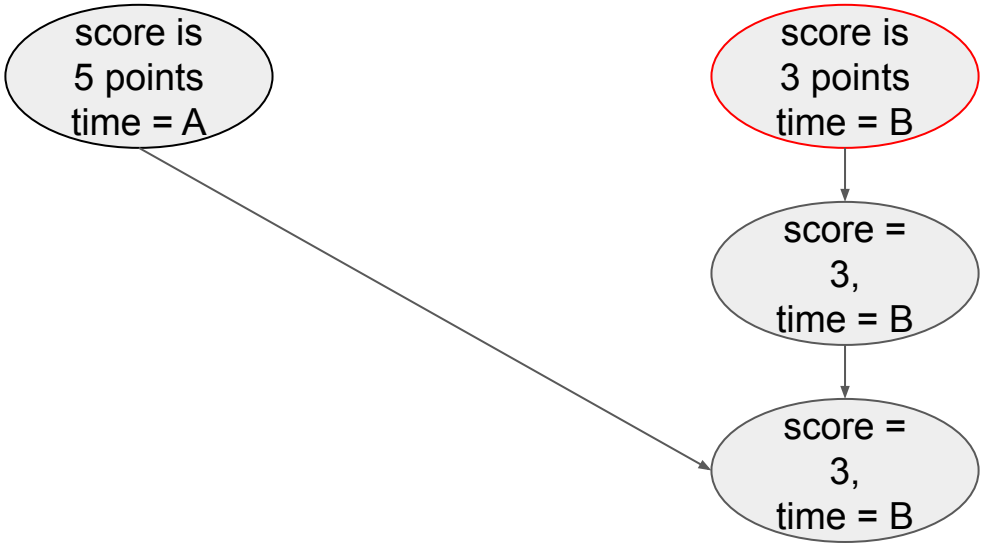
EPIC
Institute of Technology
Powered by <epam>

Solution



EPIC
Institute of Technology
Powered by epam

Solution



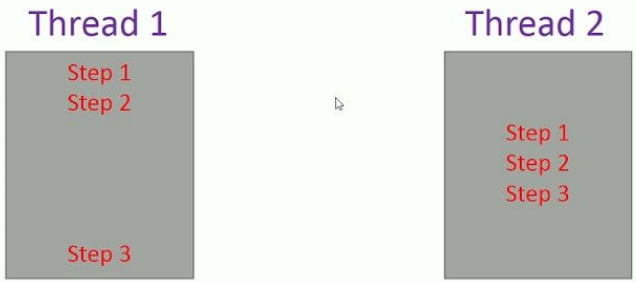
EPIC Institute of Technology
Powered by epam

Atomicity

Atomic operation : is an indivisible operation. We cannot observe such an operation half-done from any thread in the system.

i++;

- Step 1 : Read the value of 'i' from memory
- Step 2 : Increment that value by one
- Step 3 : Store the modified value to the memory



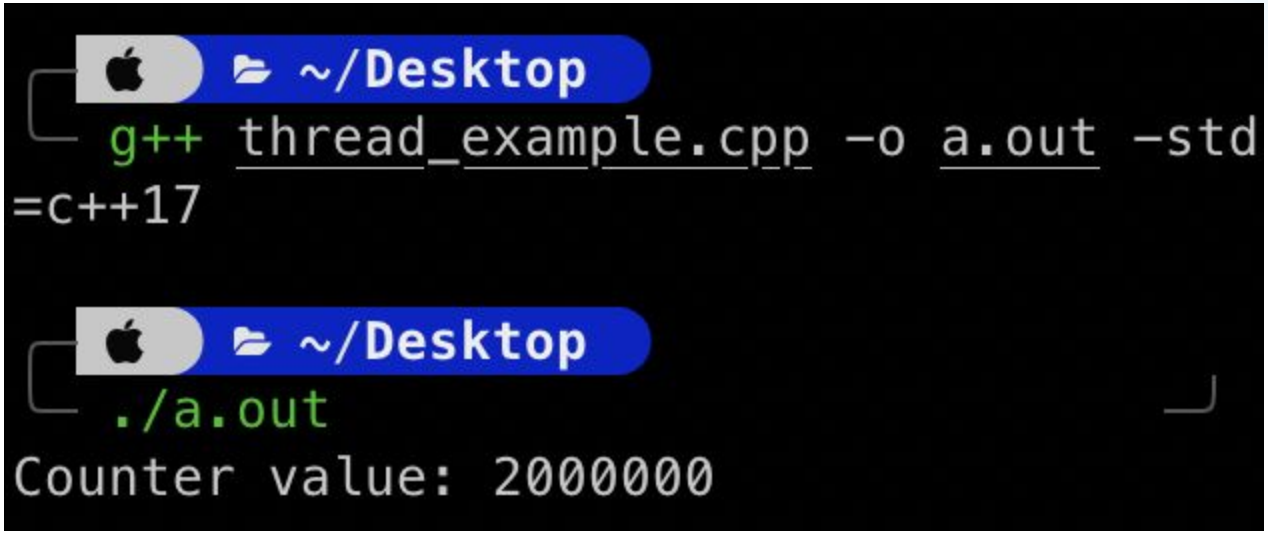
EPIC
Institute of Technology
Powered by <epam>

Atomicity

```cpp
#include <atomic>
#include <iostream>
#include <thread>

std::atomic<int> counter; // Atomic variable

void IncrementCounter()
{
    for (int i = 0; i < 1000000; ++i)
    {
        counter.fetch_add(1); // Atomic increment operation
    }
}

int main()
{
    counter = 0;

    std::thread t1(IncrementCounter);
    std::thread t2(IncrementCounter);

    t1.join();
    t2.join();

    std::cout << "Counter value: " << counter << std::endl;

    return 0;
}
```

EPIC
Institute of Technology
Powered by <epam>

Atomicity



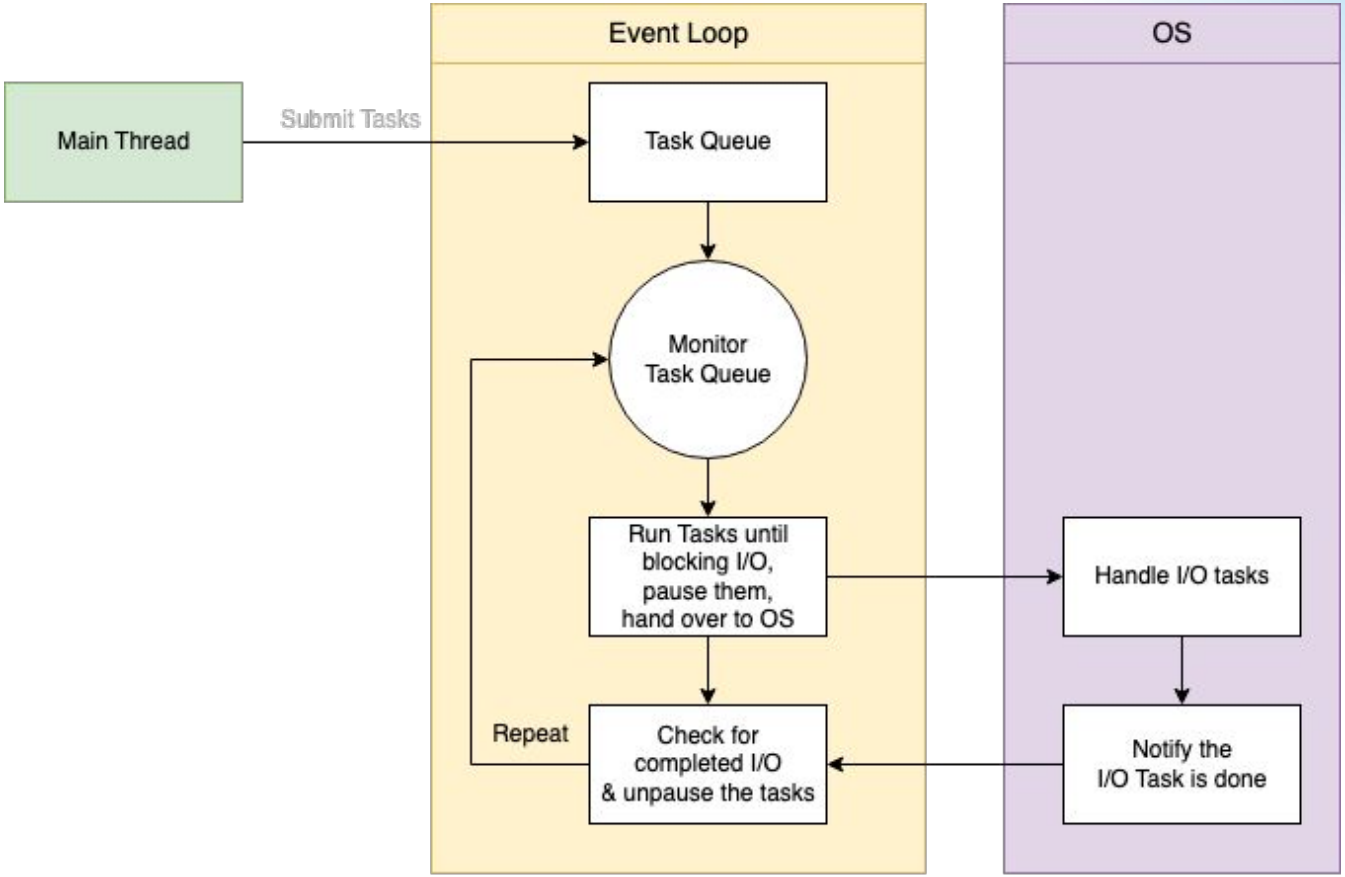
EPIC
Institute of Technology
Powered by <epam>

05

Scheduling

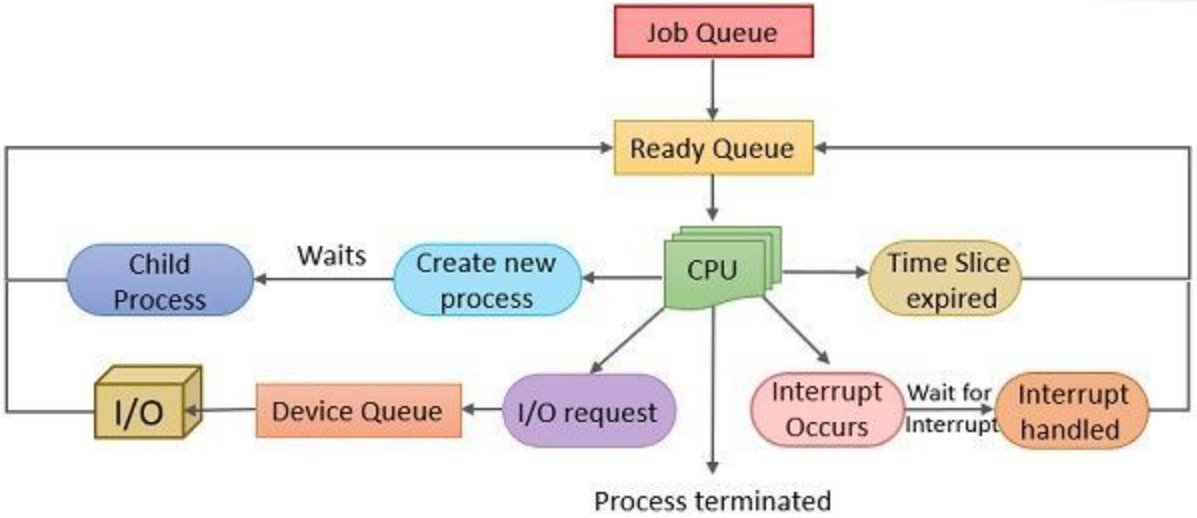
EPIC
Institute of Technology
Powered by <epam>

Sheduler



EPIC
Institute of Technology
Powered by epam

Scheduling



Scheduling Queue in Operating System

EPIC
Institute of Technology
Powered by epam

Realizations

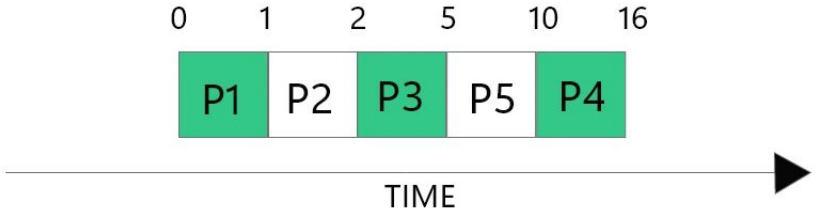
1) Round Robin
2) Priority Scheduling
3) Multilevel Queue
4) Earliest Deadline First
5) Adaptive Scheduling

EPIC Institute of Technology
Powered by epam

Priority

Priority Scheduling Algorithm

At time t=16ms

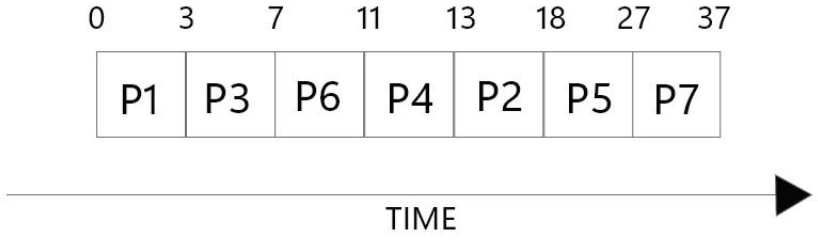


Process	P1	P2	P3	P4	P5	P6	P7
Burst Time	1	7	3	6	5	15	8
Priority	2	6	3	5	4	10	9
Arrival Time	0	1	2	3	4	5	15

EPIC Institute of Technology
Powered by epam

Priority

Priority Scheduling Algorithm



Process	P1	P2	P3	P4	P5	P6	P7
Burst Time	3	5	4	2	9	4	10
Priority	3	6	3	5	7	4	10
Arrival Time	0	2	1	4	6	5	7

EPIC Institute of Technology
Powered by <epam>

Priority Scheduling

PROS

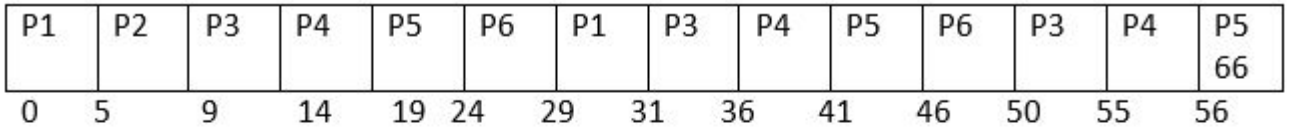
1. Higher priority - faster solution
2. Scales well

CONS

1. Starvation
2. The demands of good priorities

EPIC Institute of Technology
Powered by epam

Round-Robin



Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P1	0	7	31	31	24
P2	1	4	9	8	4
P3	2	15	55	53	38
P4	3	11	56	53	42
P5	4	20	66	62	42
P6	4	9	50	46	37

EPIC Institute of Technology
Powered by epam

Round-Robin

PROS

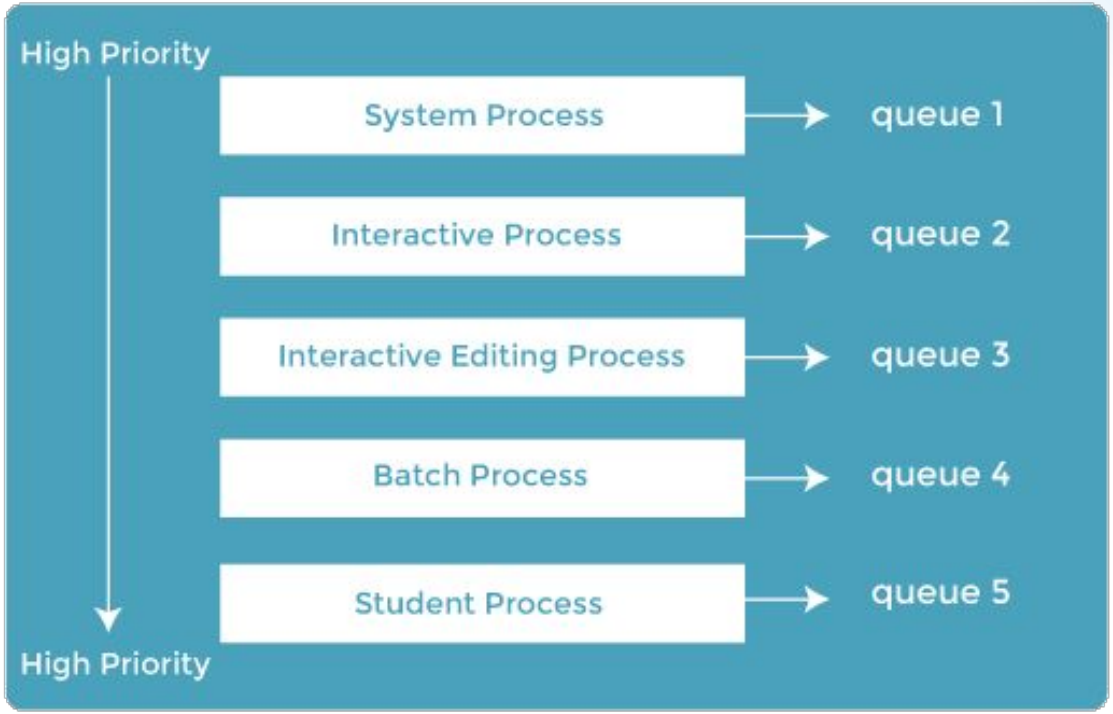
1. Fair distribution of time
2. Relatively simple implementation
3. Efficient use of CPU resources

CONS

1. Doesn't work well for different work times
2. Additional overhead for context switching between tasks

EPIC
Institute of Technology
Powered by epam

Multilevel Queue



EPIC Institute of Technology
Powered by <epam>

Multilevel Queue

PROS

1. Balance between different priorities
2. Work differentiation

CONS

1. Starvation
2. A lot of problems with the setup

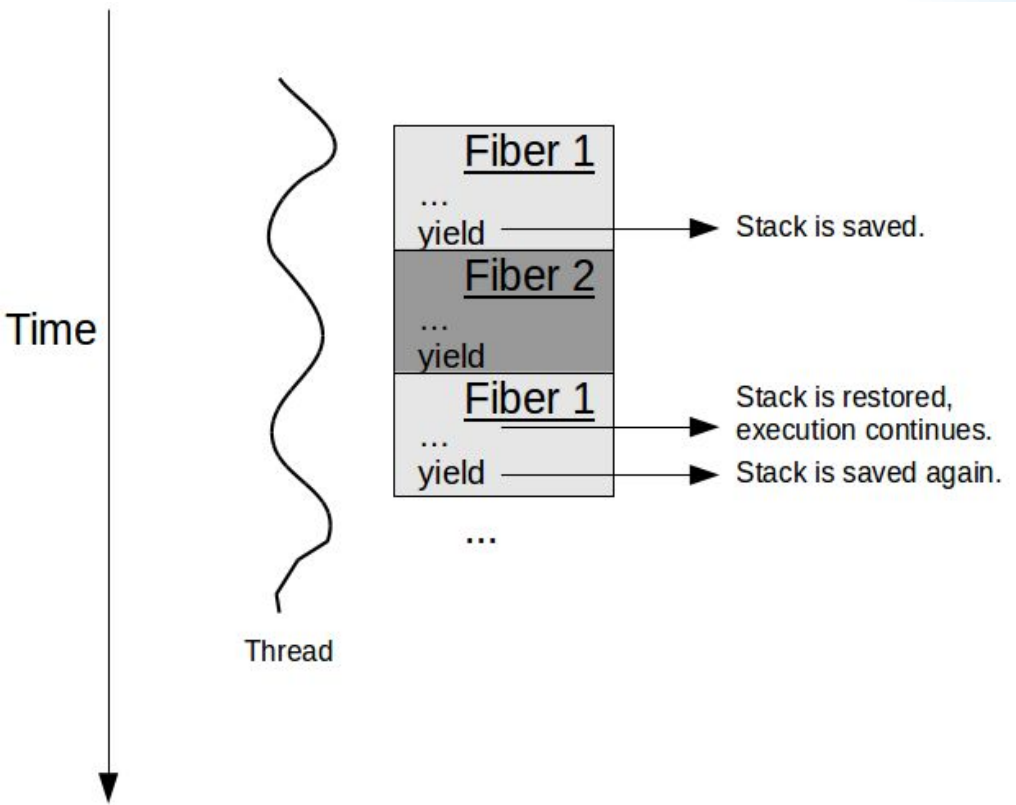
EPIC
Institute of Technology
Powered by <epam>

06

Fiber

EPIC
Institute of Technology
Powered by epam

Fiber



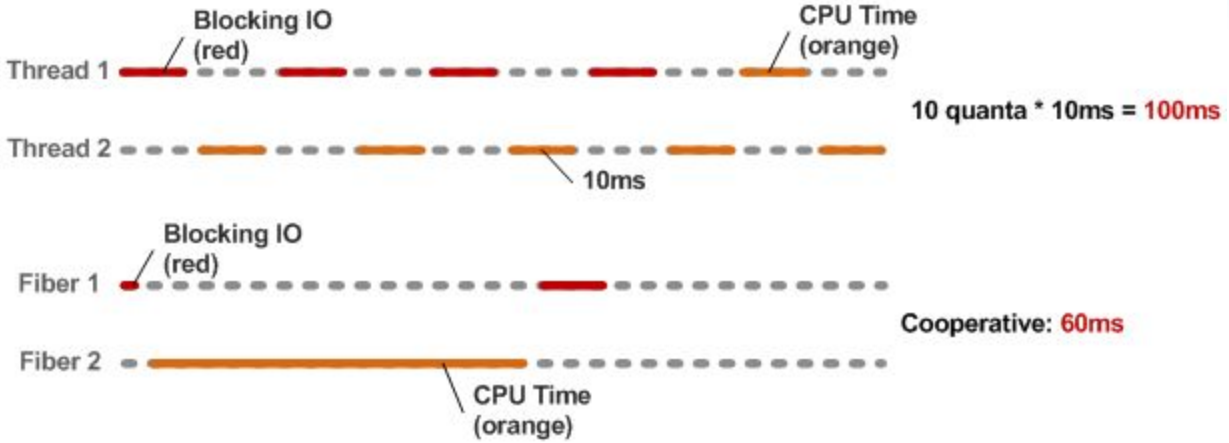
EPIC
Institute of Technology
Powered by <epam>

Why do we need fiber

1) Efficient use of CPU resources
2) Competition Management
3) Asynchronous operations
4) Parallel algorithms
5) Flexibility and control

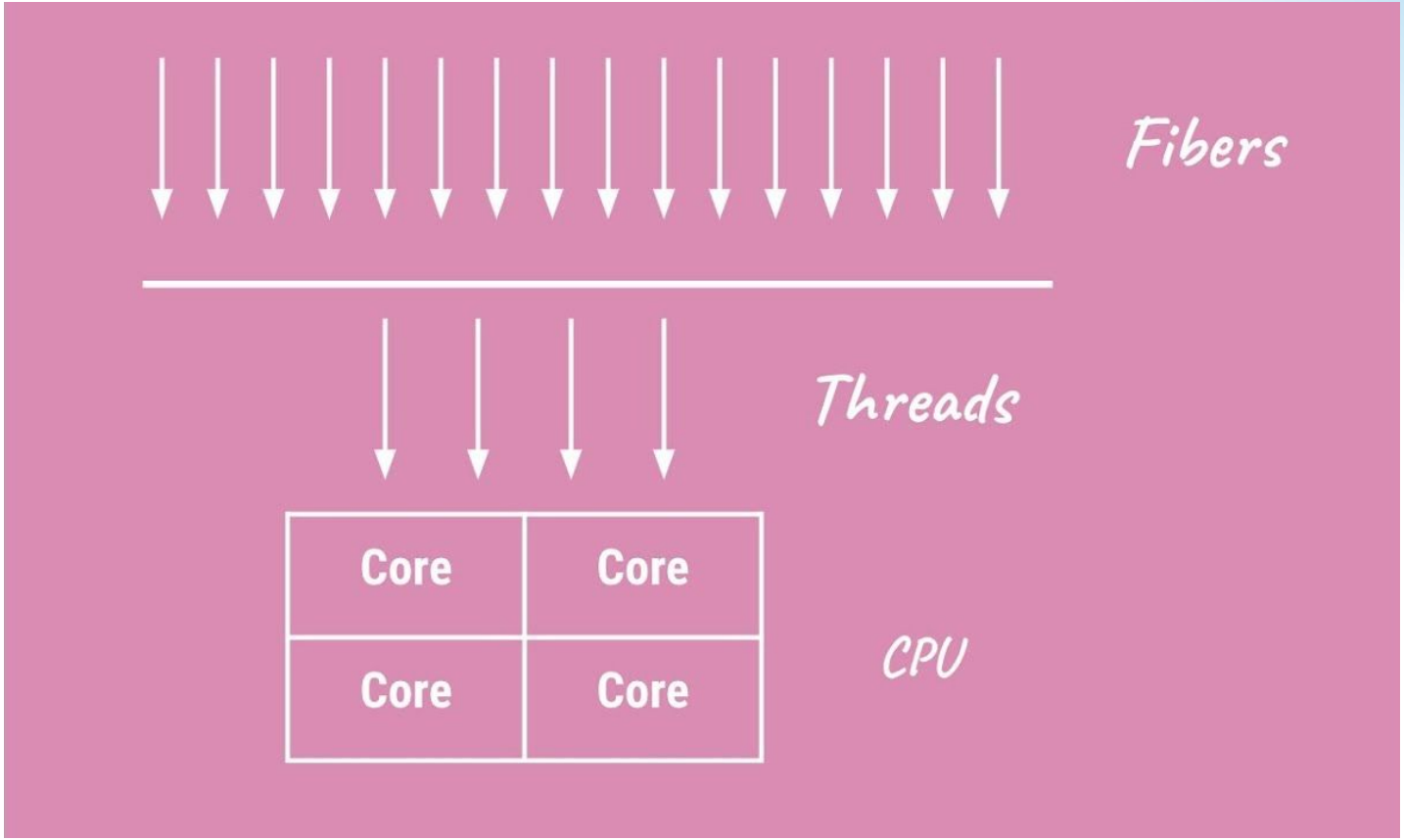
EPIC
Institute of Technology
Powered by <epam>

Fiber scheduling



EPIC
Institute of Technology
Powered by <epam>

Why fibers better



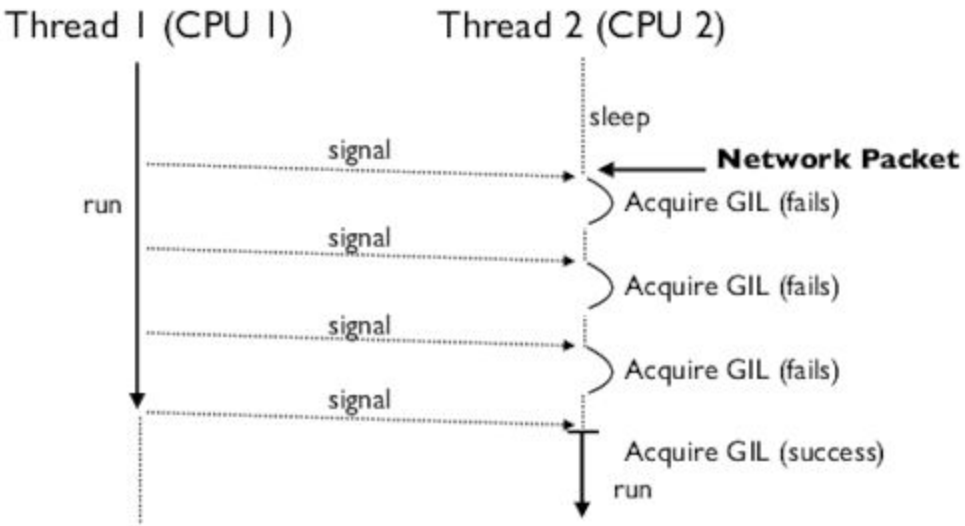
EPIC
Institute of Technology
Powered by <epam>

07

GIL

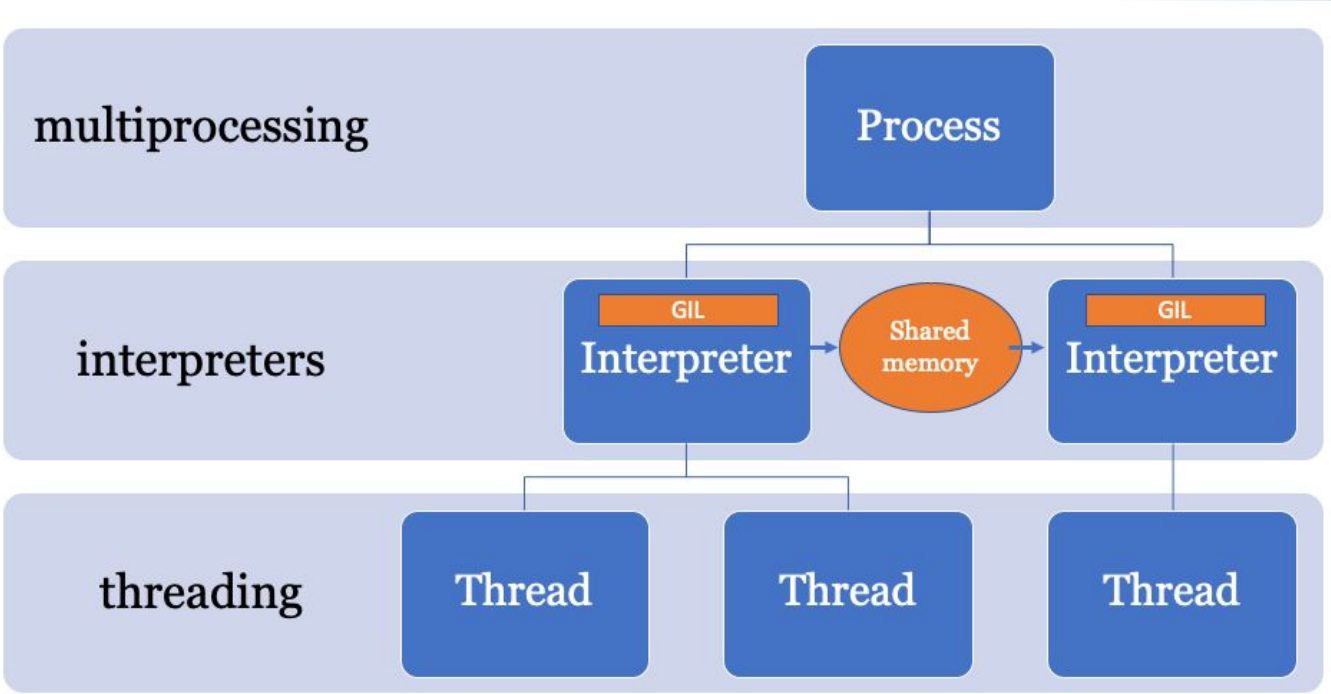
EPIC
Institute of Technology
Powered by epam

GIL



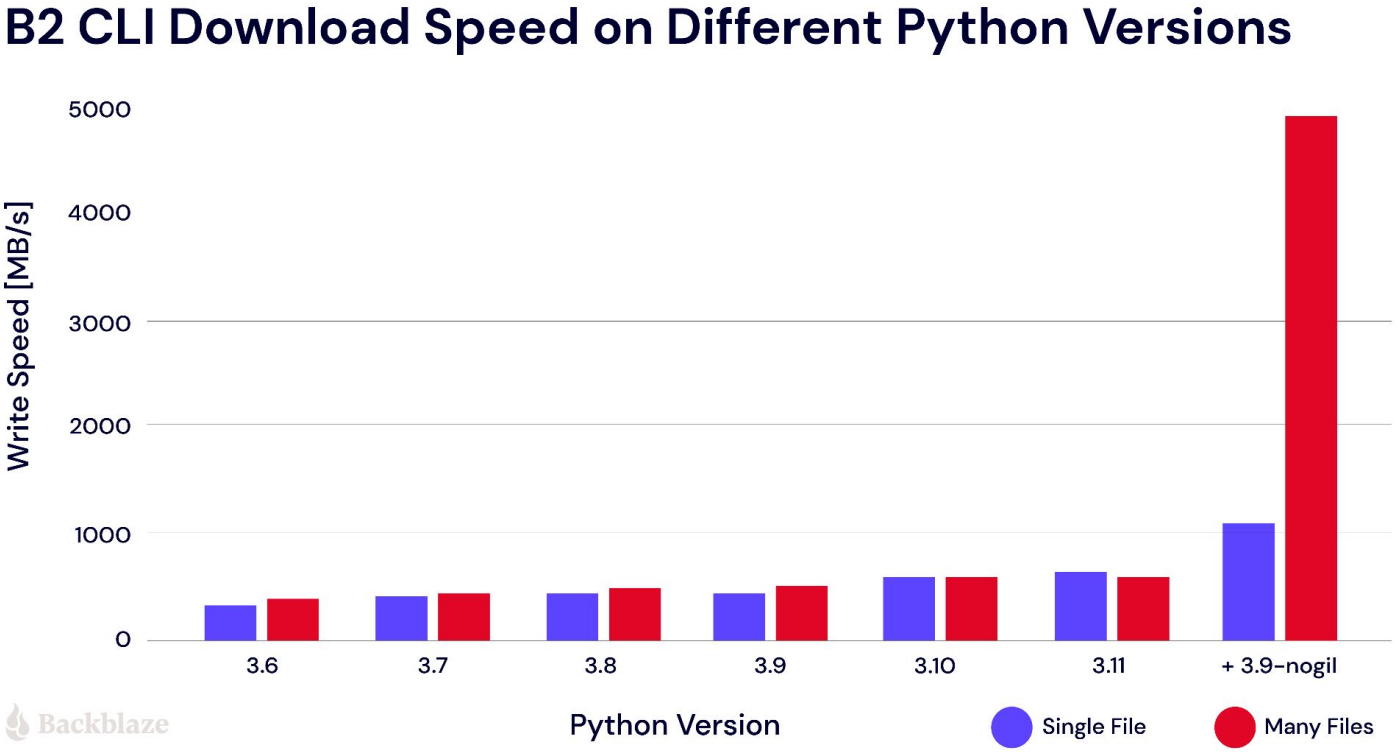
EPIC
Institute of Technology
Powered by <epam>

GIL



EPIC
Institute of Technology
Powered by epam

GIL



EPIC
Institute of Technology
Powered by <epam>

*

Good to know

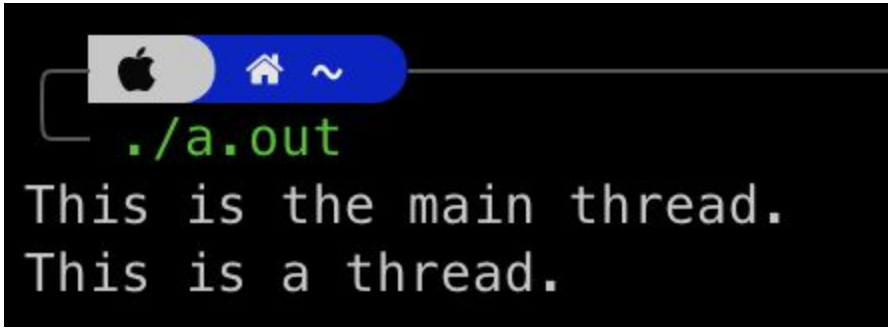
EPIC
Institute of Technology
Powered by epam

Join

```cpp
#include <iostream>
#include <thread>

// Function to be executed by the thread
void threadFunction() {
    std::cout << "This is a thread." << std::endl;
}

int main() {
    // Creating a thread object and passing it the function to execute
    std::thread myThread(threadFunction);

    std::cout << "This is the main thread." << std::endl;

    // Waiting for the thread to finish
    myThread.join();

    return 0;
}
```

EPIC
Institute of Technology
Powered by <epam>

Join



EPIC
Institute of Technology
Powered by epam

Forget join

```cpp
#include <iostream>
#include <thread>

// Function to be executed by thread
void threadFunction() {
    std::cout << "This is a thread." << std::endl;
}

int main() {
    // Creating a thread object
    std::thread myThread(threadFunction);

    std::cout << "This is the main thread" << std::endl;

    // Forget to wait
    // myThread.join();

    return 0;
}
```

EPIC
Institute of Technology
Powered by epam

Forget join

```
~/Desktop
g++ thread_example.cpp -o a.out

~/Desktop
./a.out
This is the main thread
libc++abi: terminating
This is a thread.
zsh: abort      ./a.out
```

EPIC
Institute of Technology
Powered by epam

Why?

```
~thread() _NOEXCEPT
{  // clean up
    if (joinable())
        XSTD terminate();
}
```

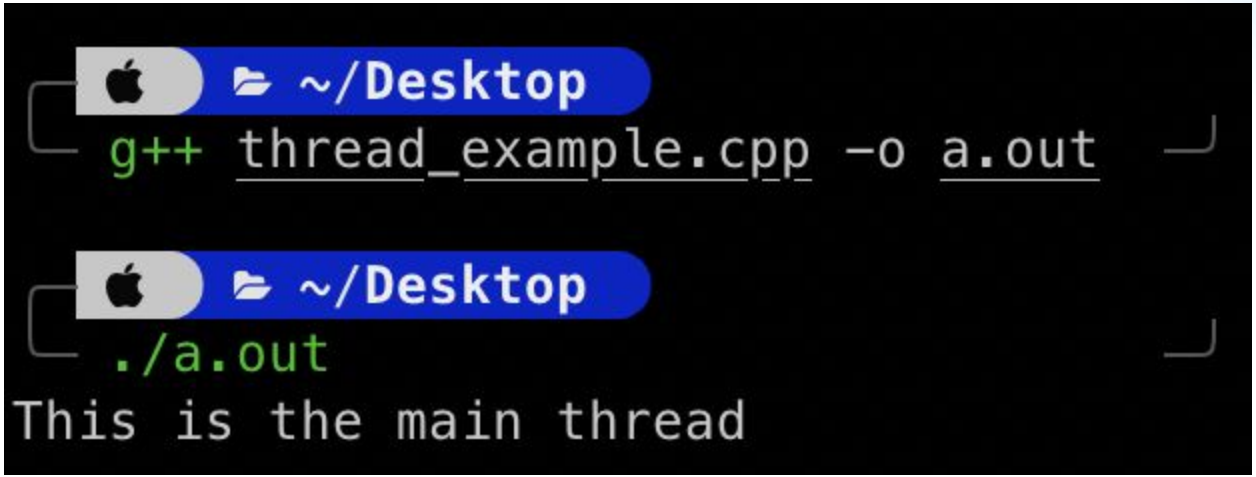
EPIC
Institute of Technology
Powered by epam

Detach

```cpp
#include <iostream>
#include <thread>

// Function to be executed by thread
void threadFunction() {
    std::cout << "This is a thread." << std::endl;
}

int main() {
    // Creating a thread object
    std::thread myThread(threadFunction);

    std::cout << "This is the main thread" << std::endl;

    // Forget to wait
    myThread.detach();

    return 0;
}
```

EPIC
Institute of Technology
Powered by <epam>

Detach



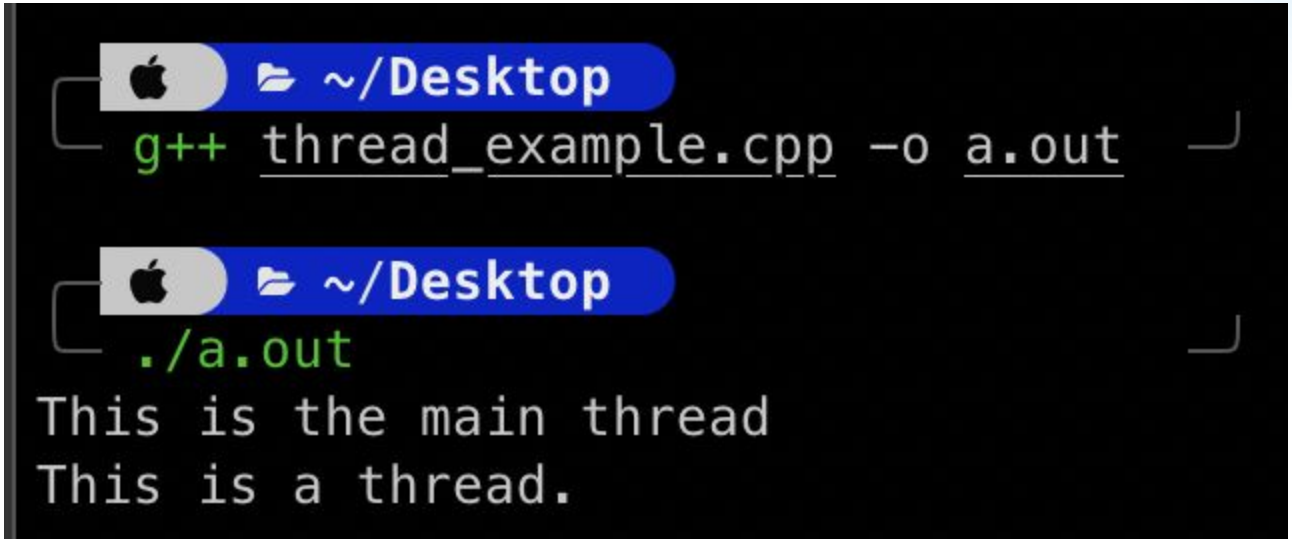
EPIC
Institute of Technology
Powered by <epam>

Detach

```cpp
#include <iostream>
#include <thread>

// Function to be executed by thread
void threadFunction() {
    std::cout << "This is a thread." << std::endl;
}

int main() {
    // Creating a thread object
    std::thread myThread(threadFunction);

    std::cout << "This is the main thread" << std::endl;

    // Forget to wait
    myThread.detach();
}
```

EPIC
Institute of Technology
Powered by <epam>

Detach



Join	Detach
1. Wait for the thread, so work longer	1. Don’t wait for the thread, so work faster
2. Wait for the thread, so work stable	2. Don’t wait for the thread, so we need to check that everything is finished

EPIC Institute of Technology
Powered by <epam>

Simple code

```cpp
#include <string>
#include <thread>
#include <iostream>
#include <functional>

using namespace std;

void ChangeCurrentMissileTarget(string& targetCity)
{
  targetCity = "Metropolis";
  cout << " Changing The Target City To " << targetCity << endl;
}

int main()
{
  string targetCity = "Star City";
  ChangeCurrentMissileTarget(targetCity);

  cout << "Current Target City is " << targetCity << endl;

  return 0;
}
```

Success #stdin #stdout 0.01s 5304KB comments (?)

stdin copy

```
Standard input is empty
```

stdout copy

```
 Changing The Target City To Metropolis
Current Target City is Metropolis
```

source

EPIC
Institute of Technology
Powered by <epam>

Multithreading

edit fork download copy

```cpp
#include <string>
#include <thread>
#include <iostream>
#include <functional>

using namespace std;

void ChangeCurrentMissileTarget(string& targetCity)
{
  targetCity = "Metropolis";
  cout << " Changing The Target City To " << targetCity << endl;
}

int main()
{
  string targetCity = "Star City";
  thread t1(ChangeCurrentMissileTarget, targetCity);
  t1.join();
  cout << "Current Target City is " << targetCity << endl;

  return 0;
}
```

Compilation error #stdin compilation error #stdout 0s 0KB comments (?)

stdin copy

```
Standard input is empty
```

compilation info

```
In file included from prog.cpp:2:
/usr/include/c++/8/thread: In instantiation of 'std::thread::thread(_Callable&&, _Args&& ...) [wit
h _Callable = void (&)(std::__cxx11::basic_string<char>&); _Args = {std::__cxx11::basic_string<cha
r, std::char_traits<char>, std::allocator<char> >&}; <template-parameter-1-3> = void]':
prog.cpp:17:51:   required from here
/usr/include/c++/8/thread:120:17: error: static assertion failed: std::thread arguments must be in
vocable after conversion to rvalues
```

[source]

EPIC Institute of Technology
Powered by <epam>

Correct

```cpp
#include <string>
#include <thread>
#include <iostream>
#include <functional>

using namespace std;

void ChangeCurrentMissileTarget(string& targetCity)
{
  targetCity = "Metropolis";
  cout << " Changing The Target City To " << targetCity << endl;
}

int main()
{
  string targetCity = "Star City";
  thread t1(ChangeCurrentMissileTarget, std::ref(targetCity));
  t1.join();
  cout << "Current Target City is " << targetCity << endl;

  return 0;
}
```

Success #stdin #stdout 0.01s 5304KB

stdin

```
Standard input is empty
```

stdout

```
 Changing The Target City To Metropolis
Current Target City is Metropolis
```

source

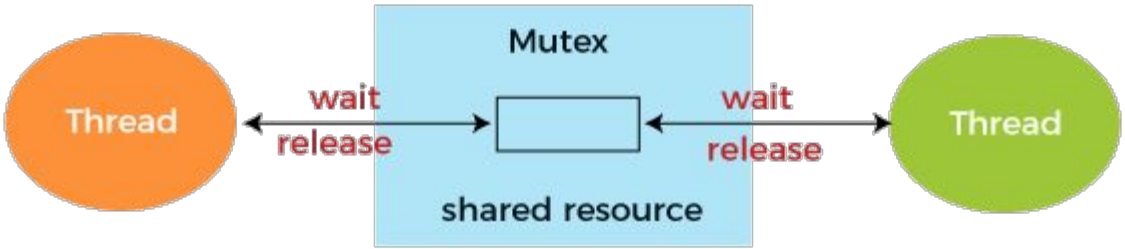
EPIC
Institute of Technology
Powered by epam

Possible reasons

1. Security
2. Easy realization
3. More independence to threads

EPIC
Institute of Technology
Powered by <epam>

Mutex



EPIC
Institute of Technology
Powered by epam

Mutex usage

```cpp
#include <mutex>
#include <thread>
#include <iostream>

std::mutex mtx; // Create a mutex object

void PrintMessage(const std::string& message)
{
    mtx.lock();
    // Critical section where the thread works with shared data
    std::cout << message << std::endl;
    mtx.unlock();
    // make something without mutex
}

int main()
{
    std::thread t1(PrintMessage, "Hello from Thread 1");
    std::thread t2(PrintMessage, "Hello from Thread 2");

    t1.join();
    t2.join();

    mtx.unlock(); // Release the mutex

    return 0;
}
```

EPIC
Institute of Technology
Powered by <epam>

Mutex usage

```
~/Desktop
g++ thread_example.cpp -o a.out -std=c++17

~/Desktop
./a.out
Hello from Thread 1
Hello from Thread 2
```

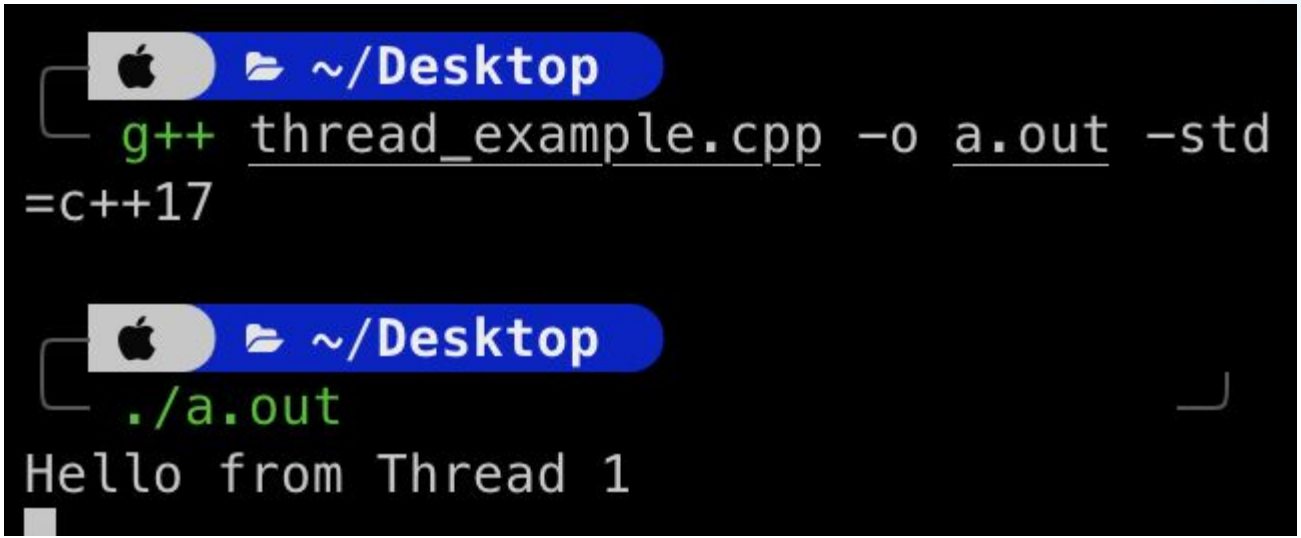
EPIC
Institute of Technology
Powered by epam

Mistake

```cpp
#include <mutex>
#include <thread>
#include <iostream>

std::mutex mtx; // Create a mutex object

void PrintMessage(const std::string& message)
{
    mtx.lock();
    // Critical section where the thread works with shared data
    std::cout << message << std::endl;
    // mtx.unlock();
    // make something without mutex
}

int main()
{
    std::thread t1(PrintMessage, "Hello from Thread 1");
    std::thread t2(PrintMessage, "Hello from Thread 2");

    t1.join();
    t2.join();

    mtx.unlock(); // Release the mutex

    return 0;
}
```

EPIC
Institute of Technology
Powered by <epam>

Mistake



EPIC
Institute of Technology
Powered by epam

With guard

```cpp
#include <mutex>
#include <thread>
#include <iostream>

std::mutex mtx; // Create a mutex object

void PrintMessage(const std::string& message)
{
    std::lock_guard<std::mutex> lock(mtx); // Acquire the mutex
    // Critical section where the thread works with shared data
    std::cout << message << std::endl;
    // The mutex is automatically released when the lock_guard goes out of scope
}

int main()
{
    std::thread t1(PrintMessage, "Hello from Thread 1");
    std::thread t2(PrintMessage, "Hello from Thread 2");

    t1.join();
    t2.join();

    return 0;
}
```

EPIC
Institute of Technology
Powered by <epam>

With guard

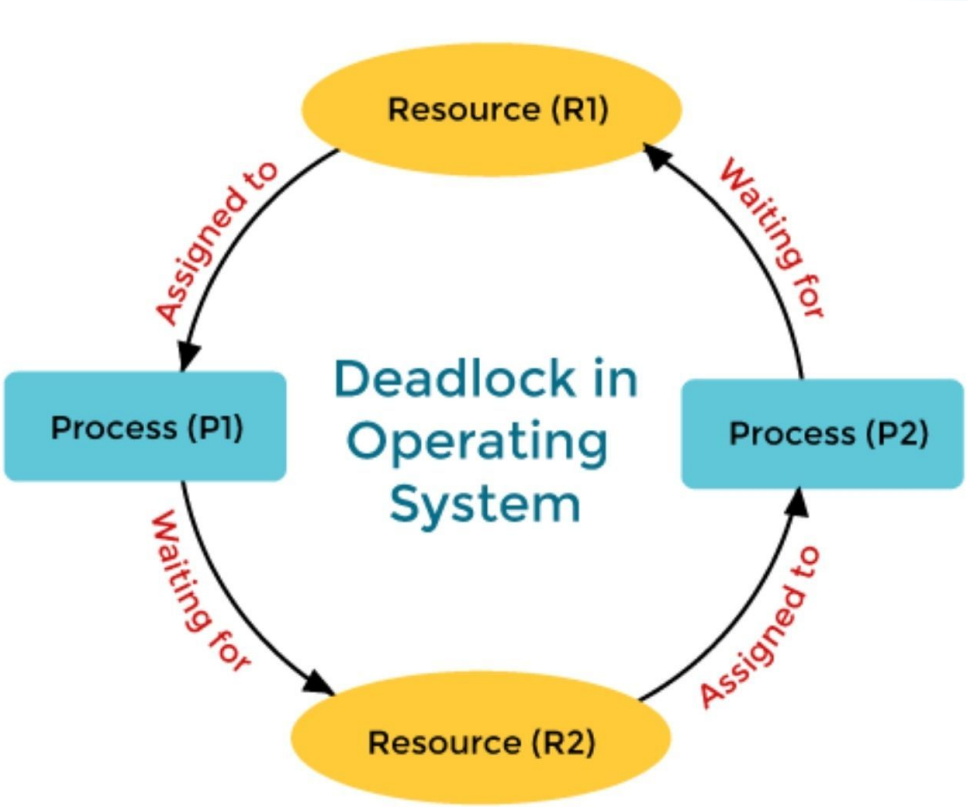
EPIC
Institute of Technology
Powered by epam

Comparison

lock/unlock	lock_guard
1. Worry about unlock 2. Can work less time because of good unlock	1. Don’t worry about unlock 2. Can work more time because of bad unlock

EPIC
Institute of Technology
Powered by epam

Deadlock



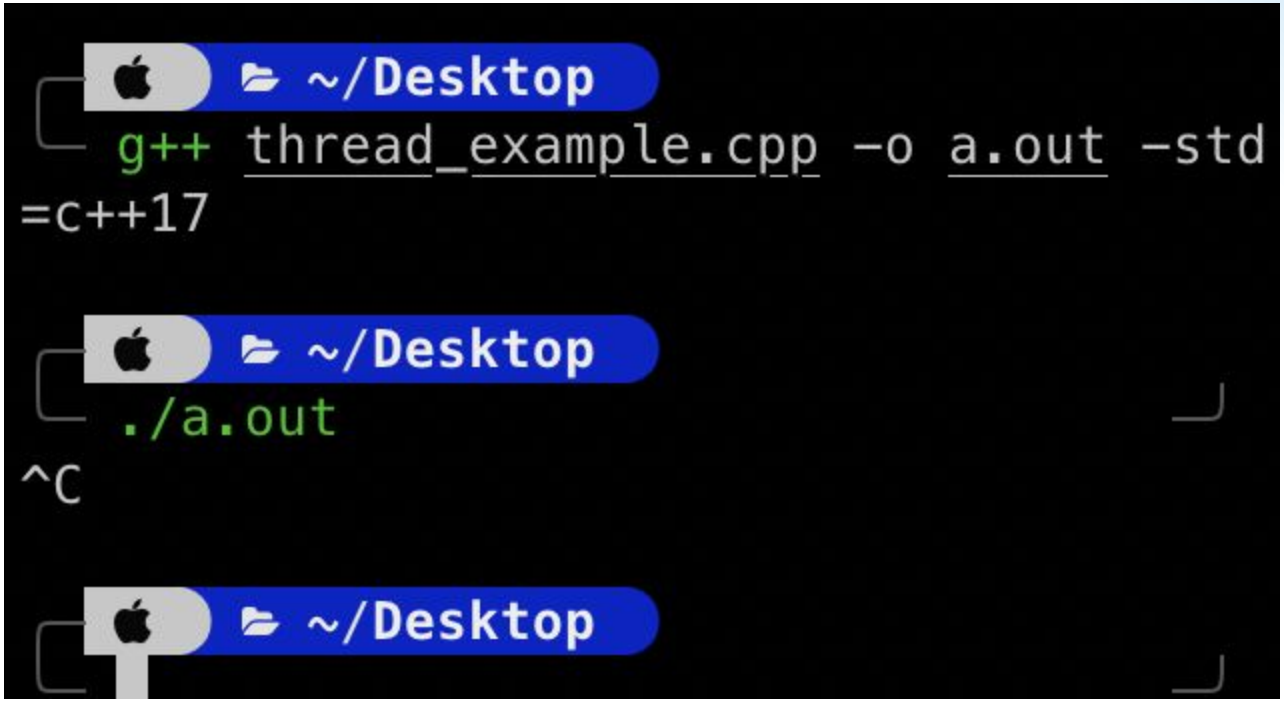
EPIC
Institute of Technology
Powered by <epam>

Deadlock

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <mutex>

using namespace std;

std::mutex muA;
std::mutex muB;

void CallHome_Th1(string message)
{
    muA.lock();
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    muB.lock();

    cout << "Thread " << this_thread::get_id() << " says " << message << endl;

    muB.unlock();
    muA.unlock();
}

void CallHome_Th2(string message)
{
    muB.lock();
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    muA.lock();

    cout << "Thread " << this_thread::get_id() << " says " << message << endl;

    muA.unlock();
    muB.unlock();
}

int main()
{
  thread t1(CallHome_Th1, "Hello from Jupiter");
  thread t2(CallHome_Th2, "Hello from Pluto");

  t1.join();
  t2.join();

  return 0;
}
```

EPIC
Institute of Technology
Powered by <epam>

Deadlock



EPIC
Institute of Technology
Powered by epam

That's All Folks!