- **Thread vs Process**
- **System Call**

Dr. Manmath N. Sahoo
Dept. of CSE, NIT Rourkela

Threads Vs Processes

```
1.  for(i=0;i<100000;i++)
2.  {
3.      receive data from i/p device   //RcvData()
4.      send data on communication link      //SendonLink()
5.  }
```

- If communication link is not free then the process will halt in line#4. Even though i/p device is free, it can be used.
- In line#3, if user has not specified any input then communication link may be underutilized.
  - If there is no other process available, then CPU can’t make a context switch even (at line#3)

Threads Vs Processes

Solution

- Consider RcvData() and SendonLink() to be two different entities
- RcvData(), on data availability, will put data in a queue
- SendonLink(), on link availability, will send data from the queue
- Whoever is free, can execute on CPU

RcvData() and SendonLink() entities can be implemented as processes or Threads

Threads Vs Processes

- Threads are the unit of execution in a process.
- A thread shares address space with it's parent.
- Per Thread items
  - Thread ID – Unique identifier
  - Program Counter – which instruction to execute next
  - Registers – for computation
  - Stack – contains the execution history
  - State – thread state

Threads Vs Processes

- Threads share the address space of the process that created it; processes have their own address space.
- Threads have direct access to the data segment of its process; processes have their own copy of the data segment of the parent process.
- Threads can directly communicate with other threads of its process; processes must use interprocess communication to communicate with sibling processes.
- Threads have almost no overhead; processes have considerable overhead.
- New threads are easily created; new processes require duplication of the parent process.

It is a light weight process and faster

Thread: Programmer’s View

```
void fn1(int arg0, int arg1, …) {…}

main() {
       …
       tid = CreateThread(fn1, arg0, arg1, …);
       …
}
```

- At the point CreateThread is called, execution continues in parent thread in main function, and execution starts at fn1 in the child thread, both in parallel

How Thread Can Help? – Example 1

- Consider the following code fragment

```
for(k = 0; k < n; k++)
      a[k] = b[k] * c[k] + d[k] * e[k];
```

- Rewrite this code fragment as:

```
CreateThread(fn, 0, n/2);
CreateThread(fn, n/2, n);
fn(l, m) {
      for(k = l; k < m; k++)
            a[k] = b[k] * c[k] + d[k] * e[k];
}
```

- What did we gain?

How Thread Can Help? – Example 2

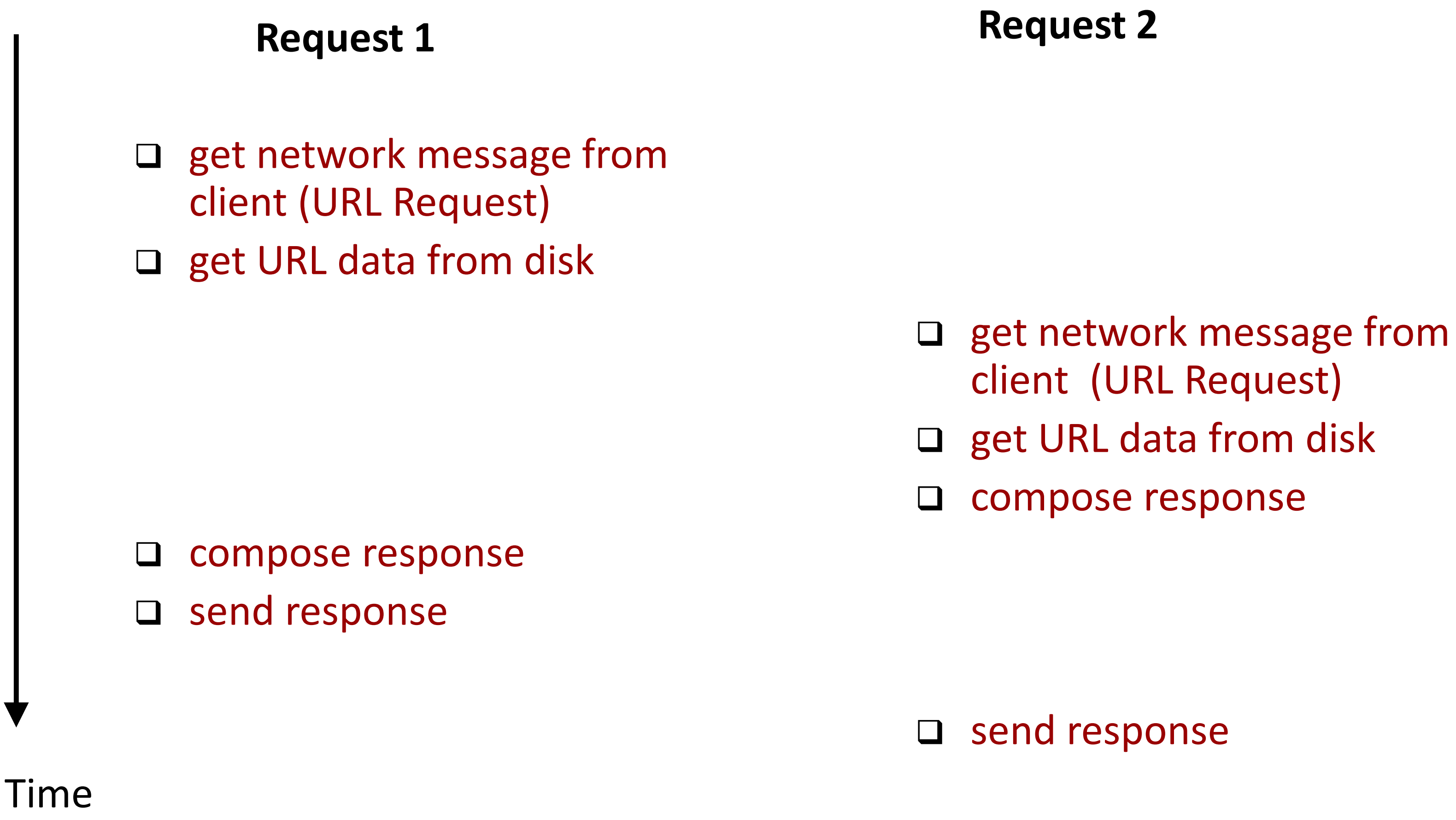
- Consider a Web server

  Create a number of threads, and for each thread do

  - get network message from client(URL Request)
  - get URL data from disk
  - compose response
  - send response

- What did we gain?

How Thread Can Help? – Example 2
Overlapping Requests (Concurrency)



Request 1

- □ get network message from client (URL Request)
- □ get URL data from disk
- □ compose response
- □ send response

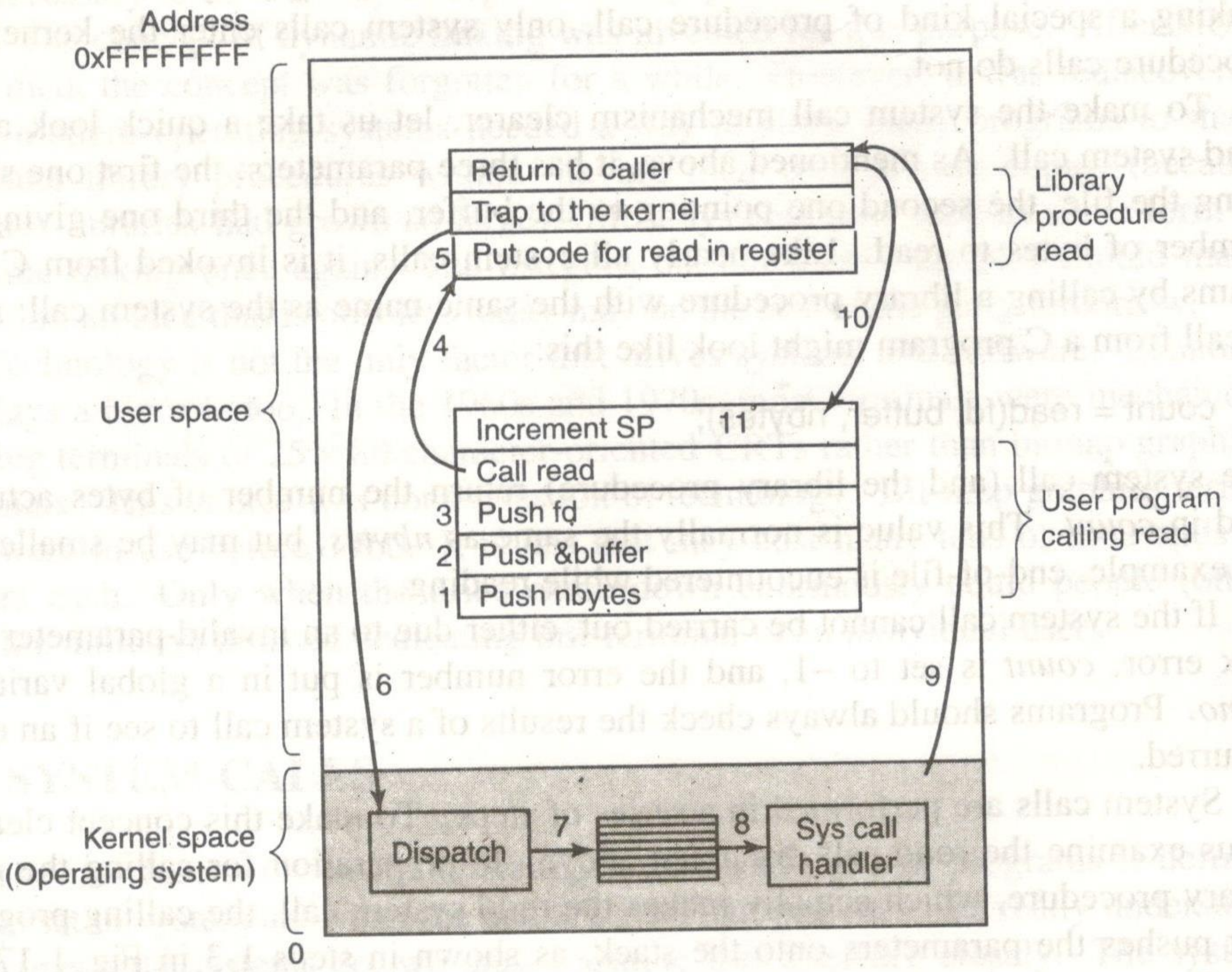
Request 2

- □ get network message from client (URL Request)
- □ get URL data from disk
- □ compose response
- □ send response

System Call

- A request by an active process to the Kernel for a service
- Defines the interface between the user and the OS
- The process switches to Kernel mode from user mode, during a system call
- Preemptive vs. non-preemptive kernel
  - NonPreemptive: Linux 2.4 Kernel
  - Preemptive: Linux 2.6 Kernel

read system call


Address
0xFFFFFFFF
Return to caller
Trap to the kernel
5 Put code for read in register
Library
procedure
read
10
4
Increment SP
11
Call read
3 Push fd
2 Push &buffer
1 Push nbytes
User program
calling read
User space
6
9
Kernel space
(Operating system)
Dispatch
7
8
Sys call
handler
0