Operating Systems 2023 Spring Term
Week 3

Dr. Emrah İnan (emrahinan@iyte.edu.tr)

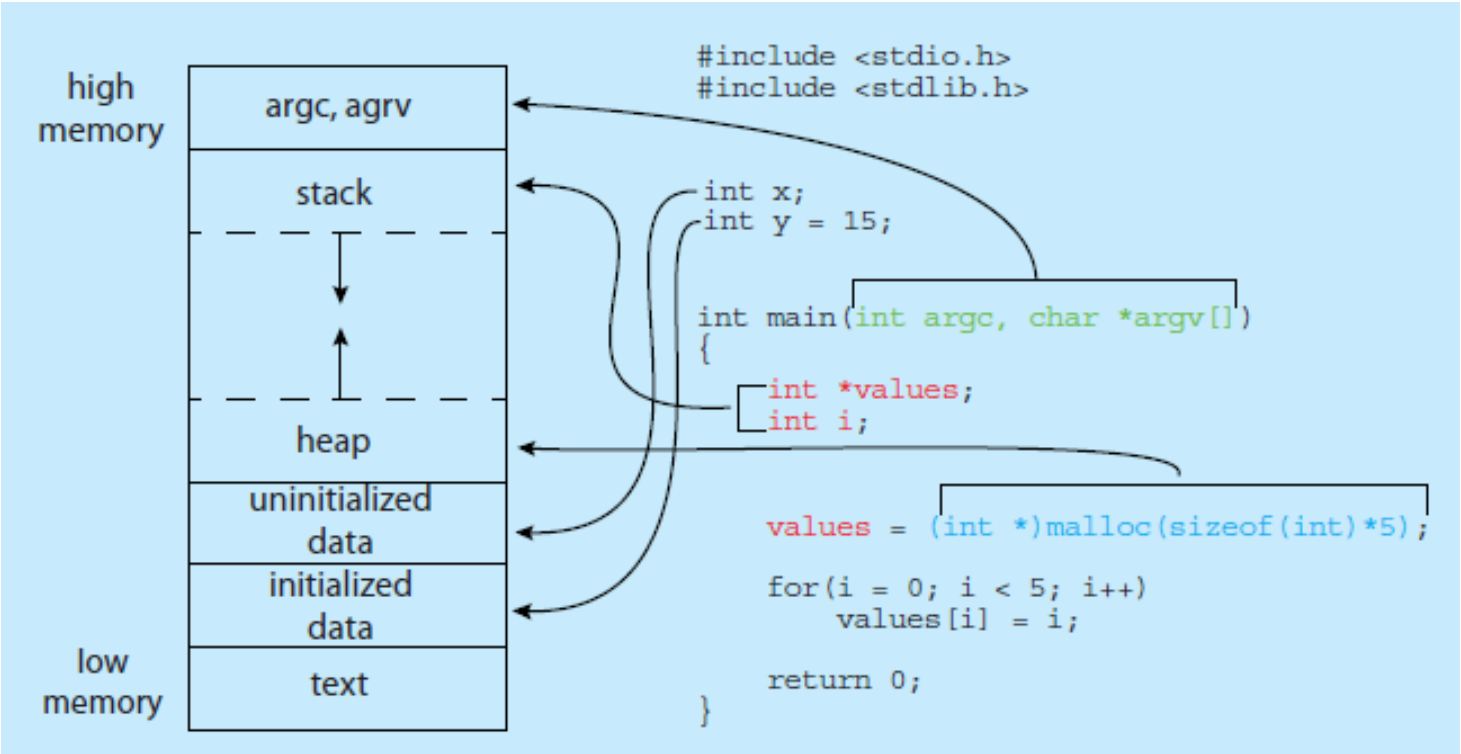
Interprocess Communication I

March 23, 2023

Week 2: Sample Glossary

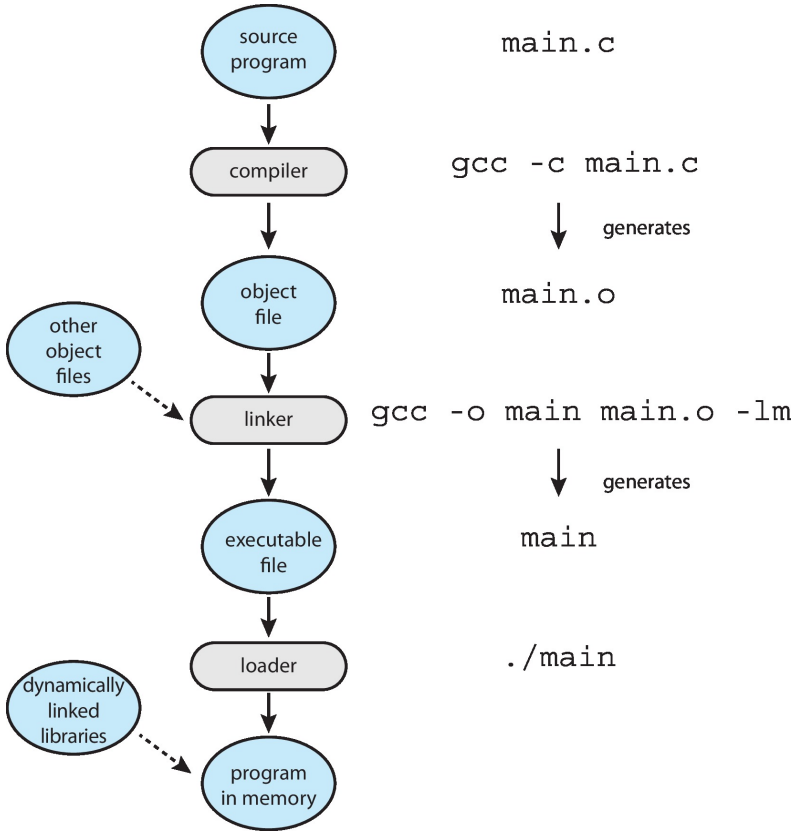
- **process:** A program loaded into memory and executing. (on Page 1263)
- **ready queue:** The set of processes ready and waiting to execute. (on Page 1265)
- **A new process is created by the fork() system call->** "The new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process. Both processes (the parent and the child) continue execution at the instruction after the fork(),with one difference: the return code for the fork() is zero for the new (child) process, whereas the (nonzero) process identifier of the child is returned to the parent." (on Page 118)
- **system call:** The primary interface between processes and the operating system, providing a means to invoke services made available by the operating system. (on Page 1271)

Memory Layout of a C Program



The Role of the Linker and Loader

Source files are compiled into object files that are designed to be loaded into any physical memory location -> relocatable object file **linker** combines these relocatable object files into a single binary executable file. **loader** is used to load the binary executable file into memory, where it is eligible to run on a CPU core.



Process-Fork Exercise I

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

int main()
{
    int pid;
    //pid_t pid; //pid struct
    pid = fork();

    if(pid > 0)
    {
        //parent
        printf("parent: %d my pid: %d\n",pid,getpid());
    }
    if(pid == 0)
    {
        //child
        printf("child: %d my pid: %d\n  ",pid,getpid());
    }
    if(pid < 0)
    {
        //error
    }
    return 0;
}
```

```
emrin@emrin:~/Downloads/cprog$ gcc cprog.c
emrin@emrin:~/Downloads/cprog$ ./a.out
parent: 5306 my pid: 5305
child: 0 my pid: 5306
```

Process-Fork Exercise I: Wait

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

int main()
{
    int pid;
    //pid_t pid; //pid struct
    pid = fork();

    if(pid > 0)
    {
        //parent
        wait(NULL); //NULL-> all children or specific pid
        printf("parent: %d my pid: %d\n",pid,getpid());
    }
    if(pid == 0)
    {
        //child
        printf("child: %d my pid: %d\n  ",pid,getpid());
    }
    if(pid < 0)
    {
        //error
    }
    return 0;
}
```

```
emrin@emrin:~/Downloads/cprog$ gcc cprog.c
cprog.c: In function ‘main’:
cprog.c:13:9: warning: implicit declaration of function ‘wait’ [-W
ion-declaration]
   13 |         wait(NULL); //NULL-> all children or specific pid
      |         ^~~~
emrin@emrin:~/Downloads/cprog$ man wait
```

Process-Fork Exercise I: man wait

```
SEE ALSO
       _exit(2), clone(2), fork(2), kill(2), ptrace(2), sigaction(2), sig-
       nal(2), wait4(2), pthread_create(3), core(5), credentials(7), signal(7)

COLOPHON
       This page is part of release 5.10 of the Linux man-pages project. A
       description of the project, information about reporting bugs, and the
       latest version of this page, can be found at
       https://www.kernel.org/doc/man-pages/.

Linux                          2020-11-01                          WAIT(2)
 Manual page wait(2) line 396/418 (END) (press h for help or q to quit)
```

```
emrin@emrin:~/Downloads/cprog$ cd /usr/include
emrin@emrin:/usr/include$ ls
aio.h              gcc-2              netiucv          stdio_ext.h
aliases.h          gconv.h            netpacket        stdio.h
alloca.h           gdb                netrom           stdlib.h
argp.h             getopt.h           netrose          string.h
argz.h             glob.h             nfs              strings.h
ar.h               gnumake.h          nl_types.h       sudo_plugin.h
arpa               gnu-versions.h     nss.h            syscall.h
asm-generic        grp.h              obstack.h        sysexits.h
assert.h           gshadow.h          openvpn          syslog.h
byteswap.h         iconv.h            paths.h          tar.h
```

Process-Fork Exercise I: grep wait *

```
threads.h:extern int cnd_timedwait (cnd_t *__restrict __cond,
threads.h:extern int __REDIRECT (cnd_timedwait, (cnd_t *__restrict __cond,
threads.h:                                __cnd_timedwait64);
threads.h:#  define cnd_timedwait __cnd_timedwait64
grep: tirpc: Is a directory
grep: video: Is a directory
wait.h:#include <sys/wait.h>
grep: X11: Is a directory
grep: x86_64-linux-gnu: Is a directory
grep: xen: Is a directory
grep: xorg: Is a directory
zlib.h:  the data provided so far to the compressor.  It may need to wait for th
e next
zlib.h:   await more bits to join them in order to fill out a full byte.  If pen
ding
```

Process-Fork Exercise I: Wait Output

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

int main()
{
    int pid;
    //pid_t pid; //pid struct
    pid = fork();

    if(pid > 0)
    {
        //parent
        wait(NULL); //NULL-> all children or specific pid
        printf("parent: %d my pid: %d\n",pid,getpid());
    }
    if(pid == 0)
    {
        //child
        printf("child: %d my pid: %d\n  ",pid,getpid());
    }
    if(pid < 0)
    {
        //error
    }
    return 0;
```

```
emrin@emrin:~/Downloads/cprog$ gcc cprog.c
emrin@emrin:~/Downloads/cprog$ ./a.out
child: 0 my pid: 7074
        parent: 7074 my pid: 7073
emrin@emrin:~/Downloads/cprog$ ./a.out
child: 0 my pid: 7237
        parent: 7237 my pid: 7236
```

Process-Fork Exercise II: How many processes created?

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();
        fork();
        fork();
        printf("hello\n");
        return 0;
}
```

```
emrin@emrin:~/Downloads/cprog$ gcc procfork.c
emrin@emrin:~/Downloads/cprog$ ./a.out
hello
hello
hello
hello
hello
hello
hello
hello
```

Process-Fork Exercise II: Tree View

```
 1 #include <stdio.h>
 2 #include <sys/types.h>
 3 #include <unistd.h>
 4 int main()
 5 {
 6         fork(); //A
 7         fork(); //B
 8         fork(); //C
 9         printf("hello\n");
10         return 0;
11 }
12         A
13        / \
14       B   B
15      / \  /\
16     C  C C  C
```

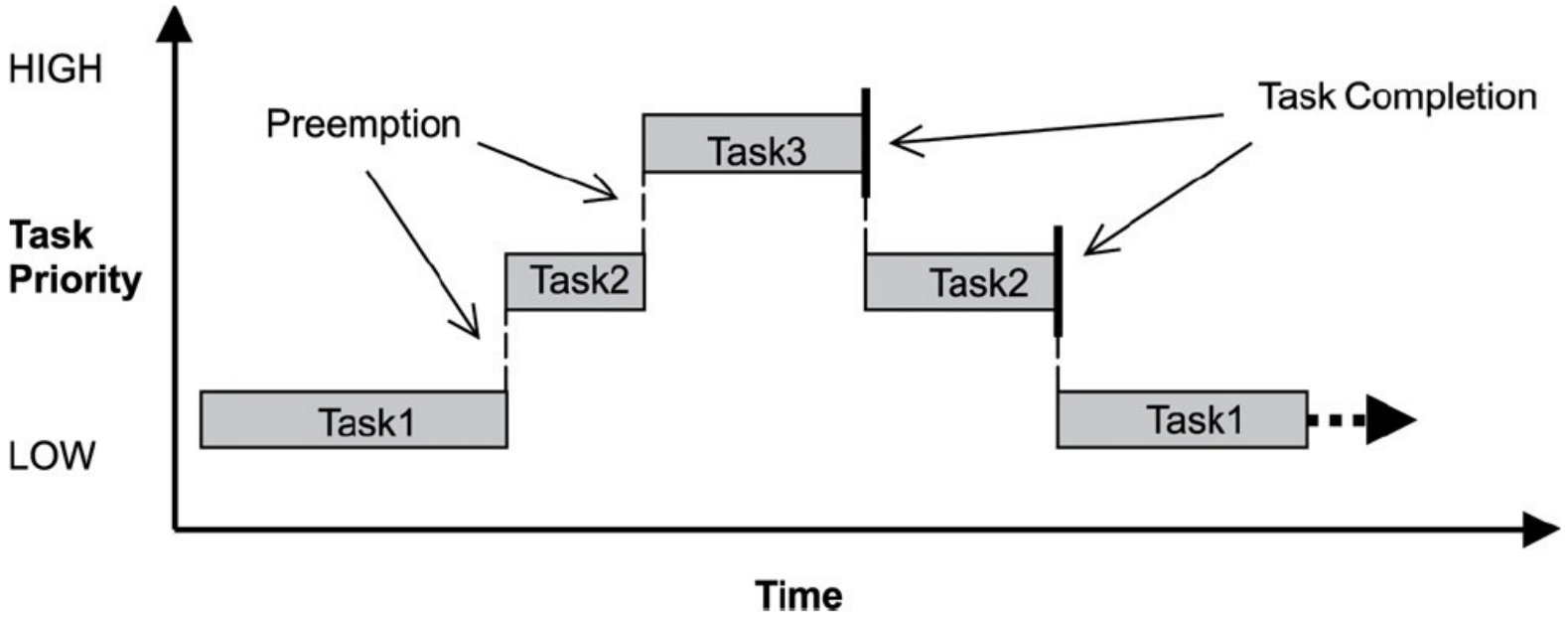
Process-Fork Exercise III: How many child processes created?

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int n = 3;
    for (int i = 0; i < n; i++){
        fork();
        printf("fork\n");
       }
return 0;
}
```

```
emrin@emrin:~/Downloads/cprog$ gcc procfork.c
emrin@emrin:~/Downloads/cprog$ ./a.out
fork
fork
fork
fork
fork
fork
fork
fork
fork
fork
fork
fork
fork
fork
```

Preemption

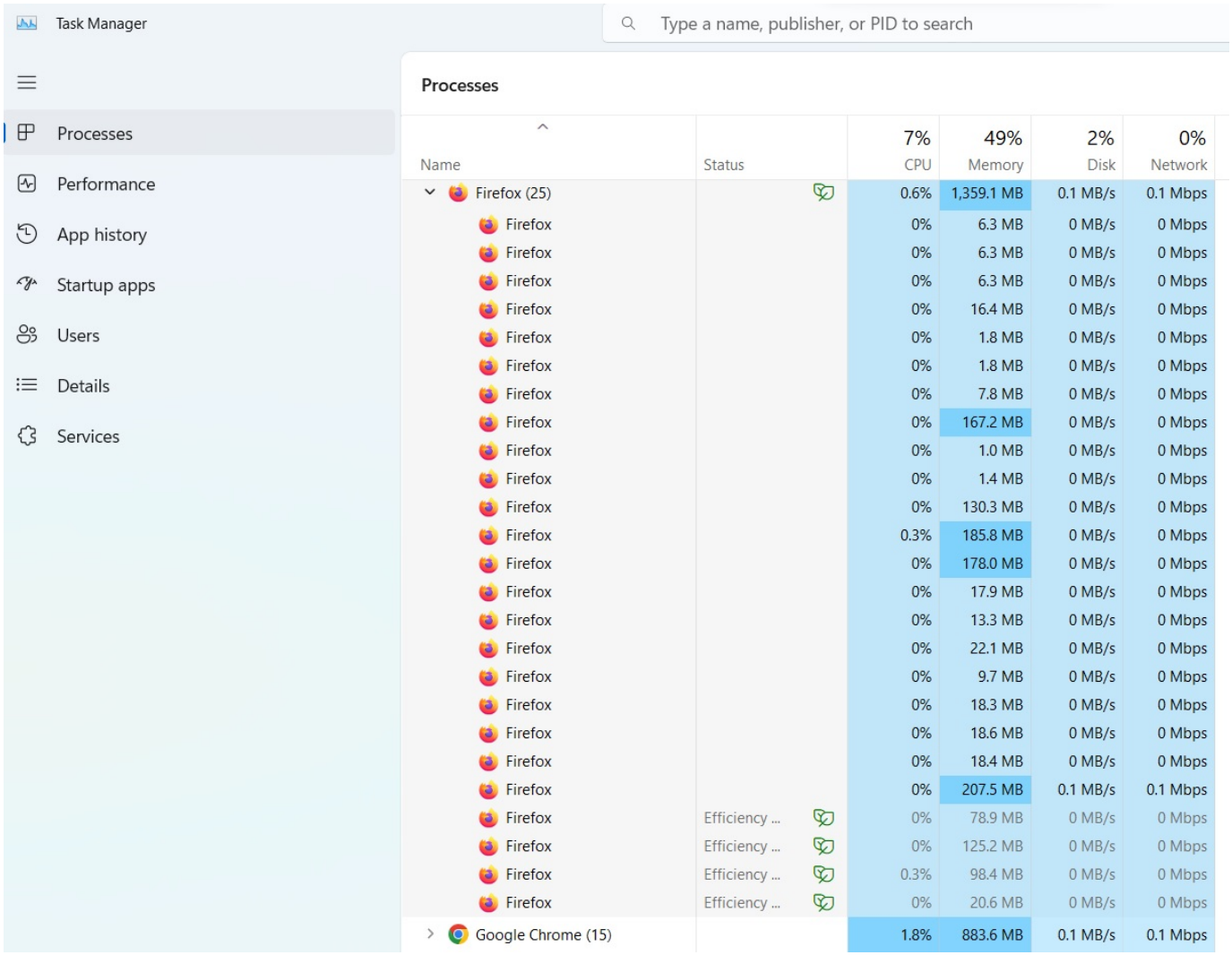
Temporarily interrupting a process with the intention of resuming the process at a later time



Dispatching

- Scheduler makes decision, then
- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

Multiprocess Architecture: Firefox vs Chrome Browser



Concurrency

- Concurrency encompasses a host of design issues, including
- (1) communication among processes,
- (2) sharing of and competing for resources (such as memory, files, and I/O access),
- (3) synchronization of the activities of multiple processes, and
- (4) allocation of processor time to processes.

Concurrency II

Concurrency arises in:

- Multiple applications -> Sharing time Multiprogramming was invented to allow processing time to be dynamically shared among a number of active applications.
- Structured applications -> Extension of modular design and structured programming, some applications can be effectively programmed as a set of concurrent processes
- Operating system structure -> OS themselves implemented as a set of processes or threads

Cooperating Processes

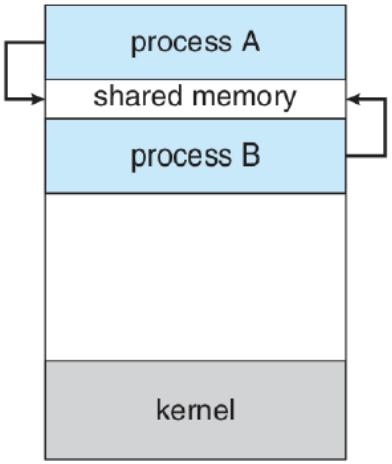
- Processes within a system may be independent or cooperating
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
- Cooperating processes need interprocess communication (IPC)

Interprocess Communication (IPC)

- Transfer data/information between address spaces
- Maintain protection and isolation
- Provide flexibility and performance
- Two models of IPC:
  - Shared memory
  - Message passing

Shared Memory IPC

- OS establishes a shared channel and maps it into each process address space
- Processes directly read/write from this memory
- OS is out of the way



Producer-Consumer Example

- A producer process produces information that is consumed by a consumer process
- For example, a compiler may produce assembly code that is consumed by an assembler
- Communication through a buffer - shared memory - of items that can be filled by the producer and emptied by the consumer
  - unbounded-buffer places no practical limit on the size of the buffer
  - bounded-buffer assumes that there is a fixed buffer size

buffer: *A memory area that stores data being transferred (e.g., between two devices or between a device and a process). (on Page 1241)*

Producer-Consumer Example: Shared Data