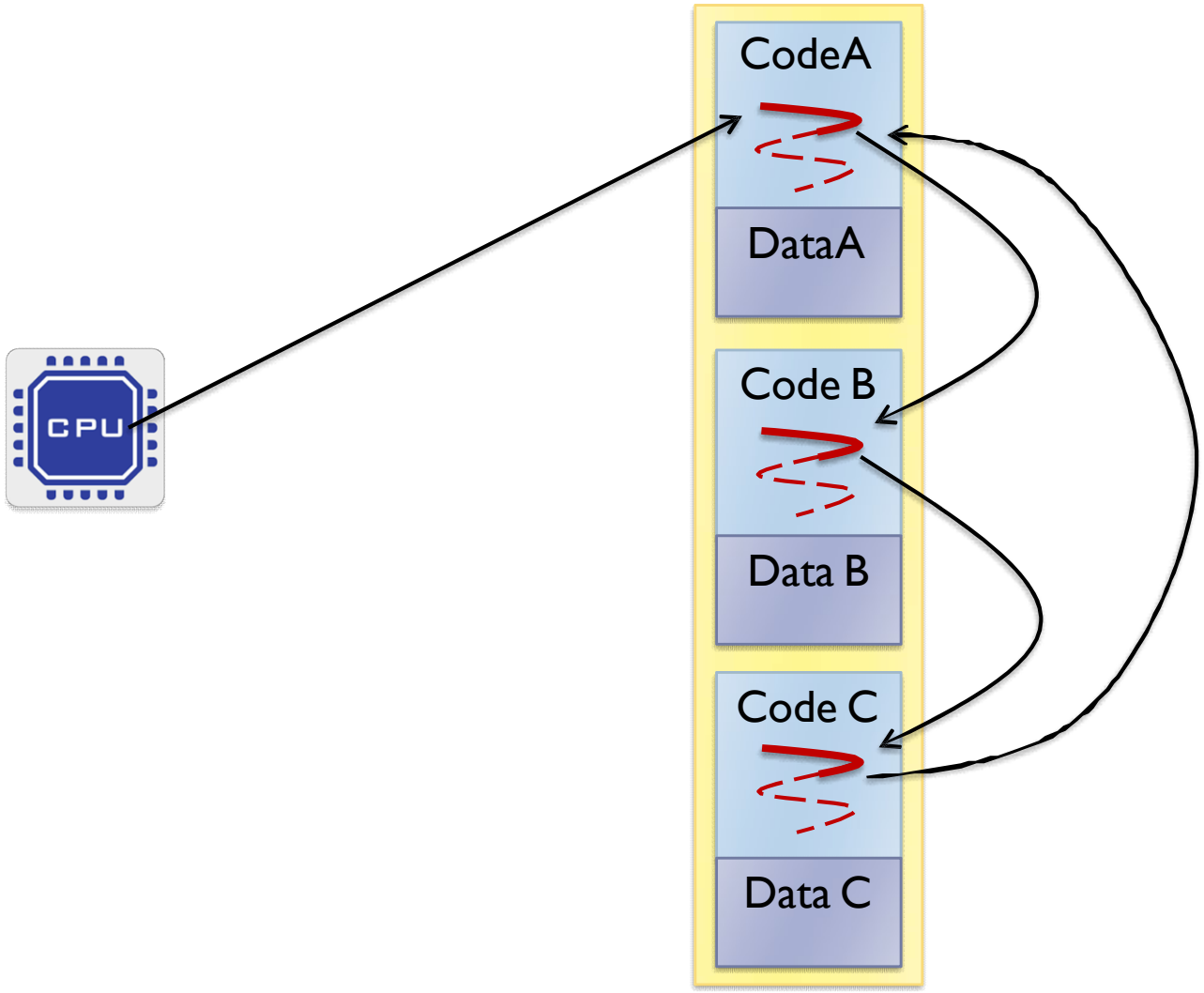
System Programming

4. Concurrent Processes

Seung-Ho Lim

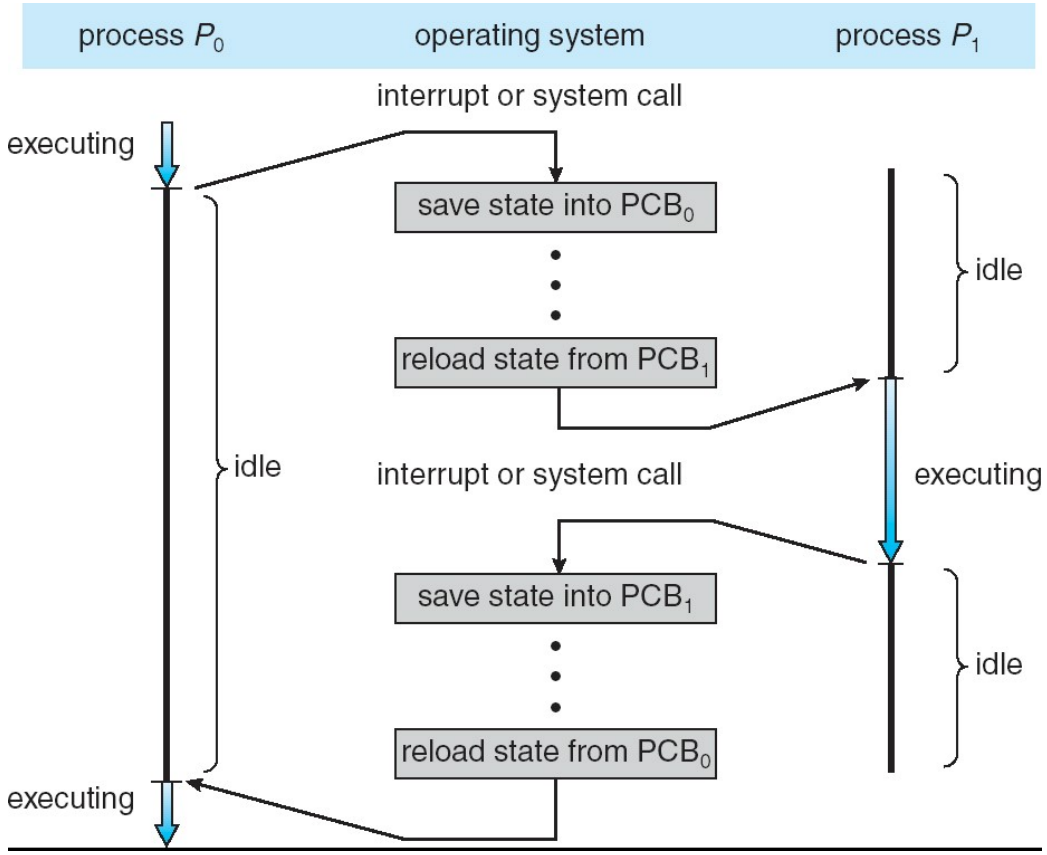
Dept. of Computer & Electronic Systems Eng.

Multitask scheduling (1)



Multitask scheduling (2)

- Context switching (or process switching)
  - happens periodically with a timer interrupt
  - OS should save current CPU context into its PCB (why?)



Multitask scheduling (3)

- Problem
  - we don't know in advance what statement in a program will be interrupted (i.e. preempted)
  - some program may result in an inconsistent output (called ***race condition***)
- Example (think what will happen in this program)

```
int main()
{   int fd1, fd2, n; char buf;
    fd1 = open ("myfile", O_RDONLY); // open in read mode
    fd2 = creat ("copyfile", 0777);  // create an empty file
    fork();
    // parent & child shares open files and their file pointers(r/w offset)
    // if a parent get a char form the file then the child get the next char
      // because they share the file pointer
    while ((n=read(fd1, &buf, 1)) != 0) write(fd2, &buf, 1);
    close(fd1); close(fd2);
}
```

File sharing b/w P&C (1)

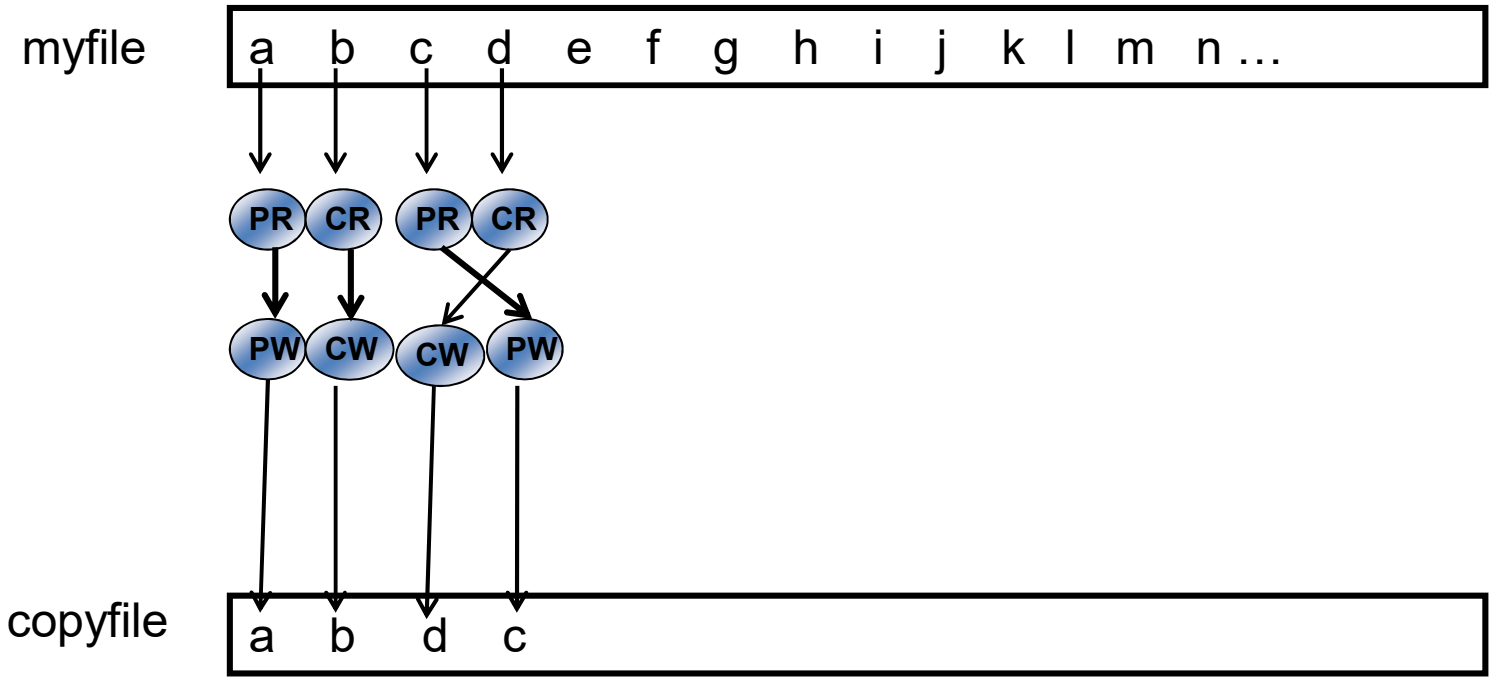
```
int main()
{
    int fd1, fd2, n; char buf;
    fd1 = open (“myfile”, O_RDONLY); // open in read mode
    fd2 = creat (“copyfile”, 0777);  // create an empty file
    fork();
    // parent & child shares open files and their file pointers(r/w offset)
    // if a parent get a char form the file then the child get the next char
     // because they share the file pointer
    while ((n=read(fd1, &buf, 1)) != 0) write(fd2, &buf, 1);
    close(fd1); close(fd2);
}
```

Context switch may occurs here !

What happens? inconsistent file critical section problem

Solution is “mutual exclusion”

File sharing b/w P&C (2)



PR: Parent's read
PW: Parent's write
CR: Child's read
CW: Child's write

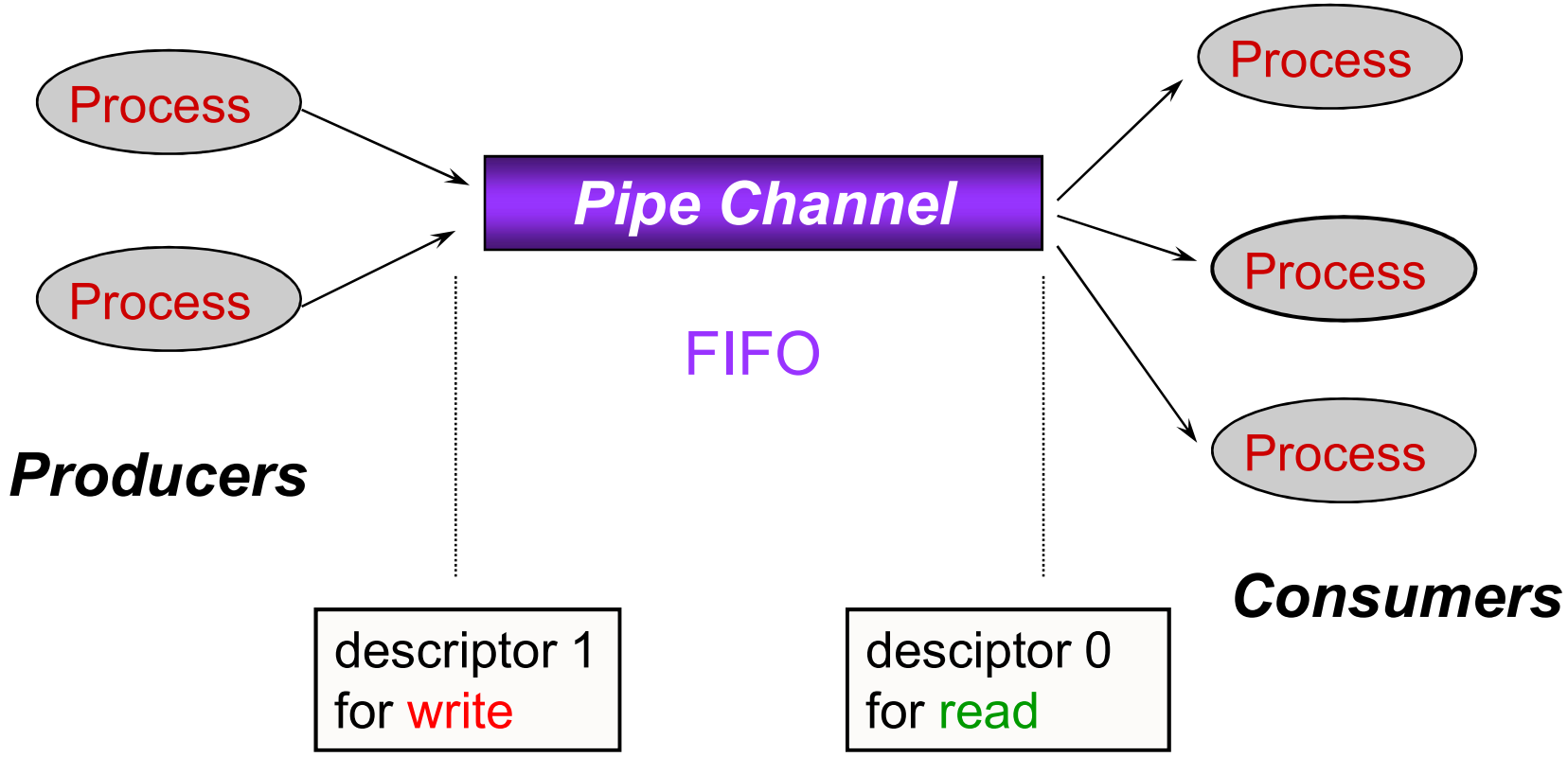
Inter-Process Communication (IPC)

- Possible methods of communications
  - By sharing files,
  - By sharing a file pointer of an opened file between parent and child
  - message queue
  - semaphores (explained later)
  - signals (ref. SP)
  - network sockets
  - ***pipe*** : circular queue bet. processes
    - named pipe : between any processes in a system
    - unnamed pipe : between parent and children

Pipe (1)

- **Pipe**
  - IPC tool between processes
  - avoid a race condition by *block & wakeup*
  - *A FIFO file created with two r/w pointers*
    - one is used for reading only
    - the other is used for writing only
  - System offered ring-buffer with variable-size queue entry and mutual exclusion;
  - Read attempt to an empty pipe will cause blocking;
  - A blocking read returns when a write occurs to the pipe or when the pipe has been closed.

Pipe (2)


Process
Process
Pipe Channel
FIFO
Process
Process
Process
Producers
Consumers
descriptor 1
for write
desciptor 0
for read

Pipe example (1)

```
int  main(void)
{    int fd[2];                    // two descripters for pipe
     int input, output;            // copy input and output
     int n; char buf[10];

     pipe(fd);      // fd[0] for reading, fd[1] for writing
     if (fork() == 0) {  // child process executes
              input = open ("input.dat", 0); // open file “input.dat” for reading
              close(fd[0]);                  // close read pipe channel
              while ( (n = read (input, buf, 10)) != 0)          // until EOF
                       write (fd[1], buf, n);                    // to pipe
              close(input);
              close(fd[1]);
              exit(0);
```

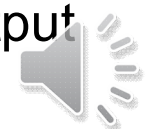
Pipe example (2)

```
    } else { // parent process executes
            close (fd[1]);      // close unnecessary wrting channel
            output = creat("output.dat", 0666);
            while ( (n = read(fd[0], buf, 10)) != 0)  // until pipe closed
                    write(output, buf, n);
            close (output);  close(fd[0]);
            wait();
    }
}
```

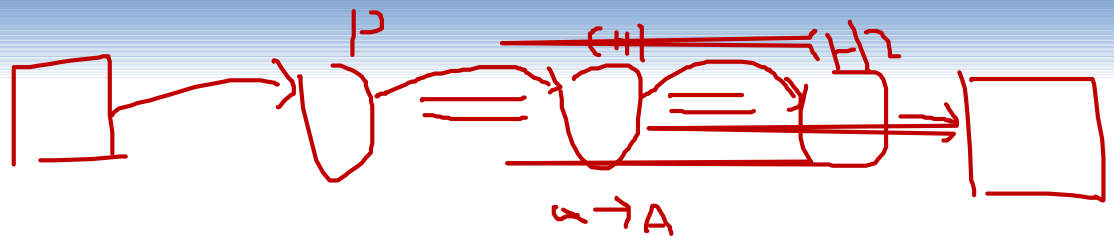
EX #1

- Fork + file sharing
  1. Open a text file, and then fork 2 times (3 processes) to share the file,
  2. Each process (including the parent process) read from the text file with while loop until EOF, for each read
     1. reads 10 characters from the file
     2. then prints its pid & the character
     3. Sleep(1)
  3. At EOF, children processes exit and the parent process waits for the deaths of all children.

- $> ./a.out input-file > output-file
- Check the contents & the order of characters of the input & output file.



EX #2



- Fork + IPC
  1. Open a text file, create an output file; create two pipes. (pipe 1 and 2)
  2. Fork two children;
  3. Parent: read 10 lower case letters (10 bytes) and them put them into pipe1 until EOF;
  4. Child 1: get 10 bytes from pipe 1 and convert them into capital letters; and them put them into pipe 2 until pipe1 closed;
  5. Child 2: get 10 bytes from pipe 2 and write them into the output file until pipe2 closed;
  6. Parent process waits for deaths of children.

  - Check the contents of output file.

HW

- Due date
  - 4/16(Thursday)
- 제출 방법
  - EX#1, EX#2 소스코드 작성
  - 소스코드 파일
    - 서버에 HW2 디렉토리를 생성하고 세가지 소스코드를 작성
  - 보고서
    - 소스코드 설명과 실행 결과를 보고서 형식으로 작성하여 Word, hwp, pdf파일로 eclass에 upload