UPPSALA UNIVERSITET

Informationsteknologi

Today's class

- Review of more C
- Operating system overview

Review of more C

UPPSALA UNIVERSITET

Informationsteknologi

File handling

- Open a file using “fopen”
- Returns a file pointer which is used to access the file
- Modes
  - Read(r) – error if file does not already exist. File pointer at the beginning of file.
  - Write(w) – create a new file (overwrite old one). File pointer at the beginning of file.
  - Append(a) – create a new file if file does not exist. Preserve the contents if file does exist and pointer at the end of the file.
- fprintf, fscanf, fclose

UPPSALA UNIVERSITET

Informationsteknologi

Example 8

```
#include <stdio.h>

int
main(int argc, char *argv[])
{
 FILE *inFile=NULL; /* Declare a file pointer */

 inFile = fopen("test.txt", "w"); /* open file for writing*/

 if(inFile == NULL){ /* need to do explicit ERROR CHECKING */
    exit(1);
  }
   /* write some data into the file */
    fprintf(inFile, "Hello there");

   /* don't forget to release file pointer */
    fclose(inFile);

    return 0;
}
```

Reading until end of file

- int feof(FILE *) – The function is defined in stdio.h
  - Returns a non-zero value if end of file has been reached, and zero otherwise.

Sample code:

```
fscanf(inFile, "%d", &int1); // Try to read
while (feof(inFile) == 0 ){  //If there is data, enter loop
    printf("%d \n", int1);    //Do something with the data
    fscanf(inFile, "%d", &int1);  //Try reading again
} //go back to while to test if data was read
```

Functions – why and how

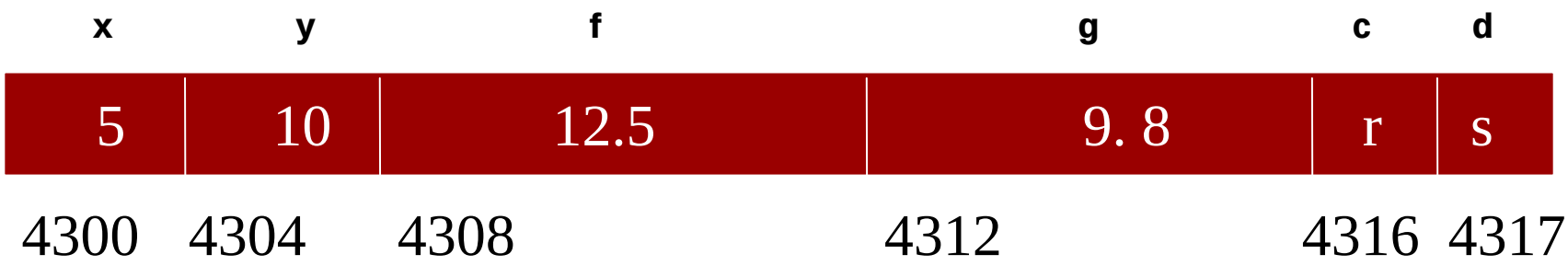
- If a problem is large
- Modularization – easier to
  - code
  - debug
- Code reuse
- Passing arguments to functions
  - By value
  - By reference
- Returning values from functions
  - By value
  - By reference

Functions – basic example

```
Example 9

#include <stdio.h>
int sum(int a, int b);
        /* function prototype at start of file */

int
main(int argc, char *argv[])
{
   int total = sum(4,5); /* call to the function */

   printf("The sum of 4 and 5 is %d\n", total);
}

int sum(int a, int b){   /* the function itself
                          - arguments passed by value*/
   return (a+b);       /* return by value */
}
```

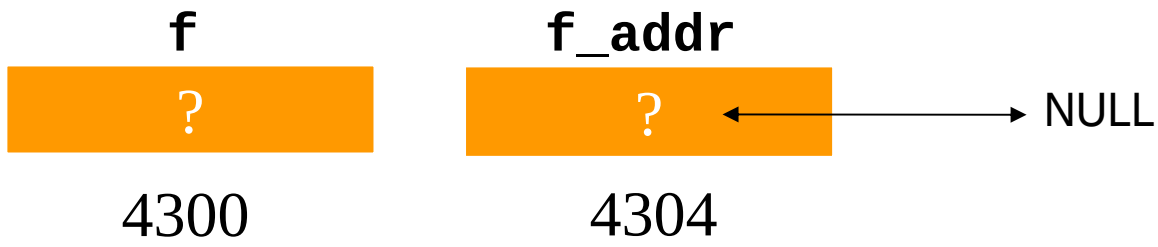
Memory layout and addresses

```
int x = 5, y = 10;
float f = 12.5, g = 9.8;
char c = ‘r’, d = ‘s’;
```

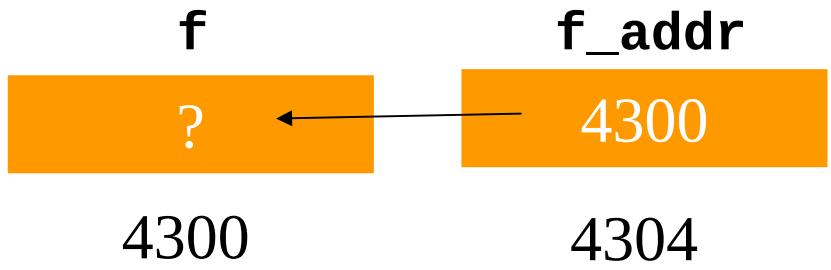


Pointers made easy

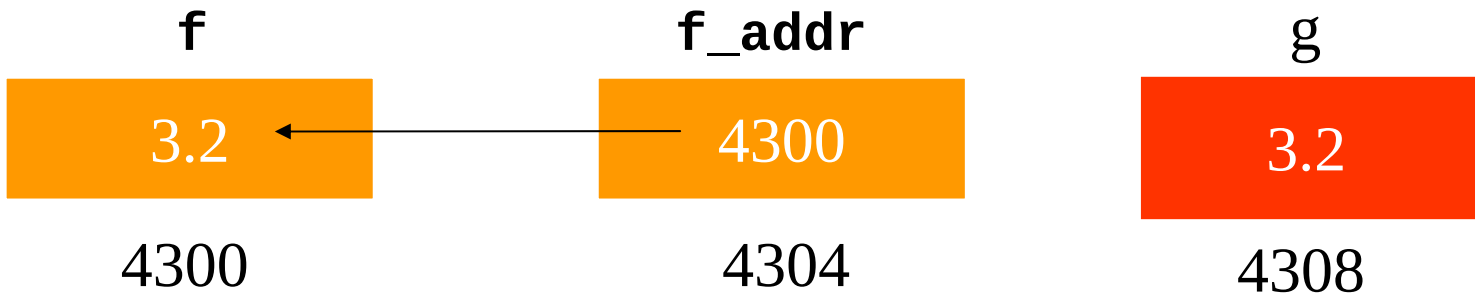
```
float f;          // data variable  - holds a float
float *f_addr;    // pointer variable – holds an address to
                  // a float
```



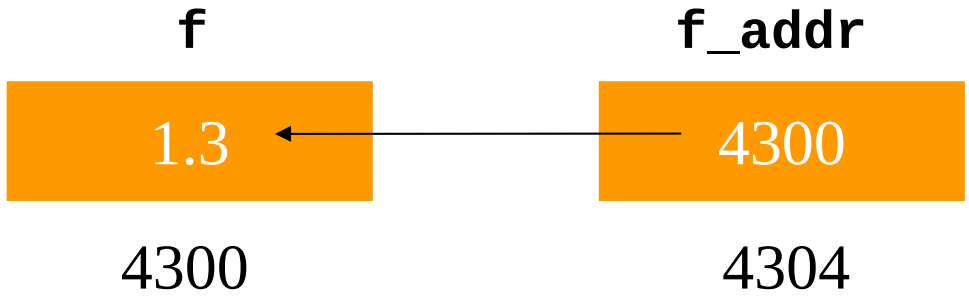

```
f_addr = &f;      // & = address operator
```



```
*f_addr = 3.2;   // indirection operator or dereferencing
float g=*f_addr; // indirection: g is now 3.2
```




```
f = 1.3;
```



Pointer operations

- Creation
  - `int *ptr;`
- Pointer assignment/initialization
  - `ptr = &i;` (where i is an int and &i is the address of i)
  - `ptr = iPtr;` (where iPtr is a pointer to an int)
- Pointer indirection or dereferencing
  - `i = *ptr;` (i is an int and *ptr is the int value pointed to by ptr)

Example 10

```
#include <stdio.h>

int
main(int argc, char *argv[])
{
        int j;
        int *ptr;

        ptr=&j;   /* initialize ptr before using it */
                  /* *ptr=4 does NOT initialize ptr */

        *ptr=4;   /* j <- 4 */

        j=*ptr+1; /* j <- ??? */

        return 0;
}
```

Pointers and arrays

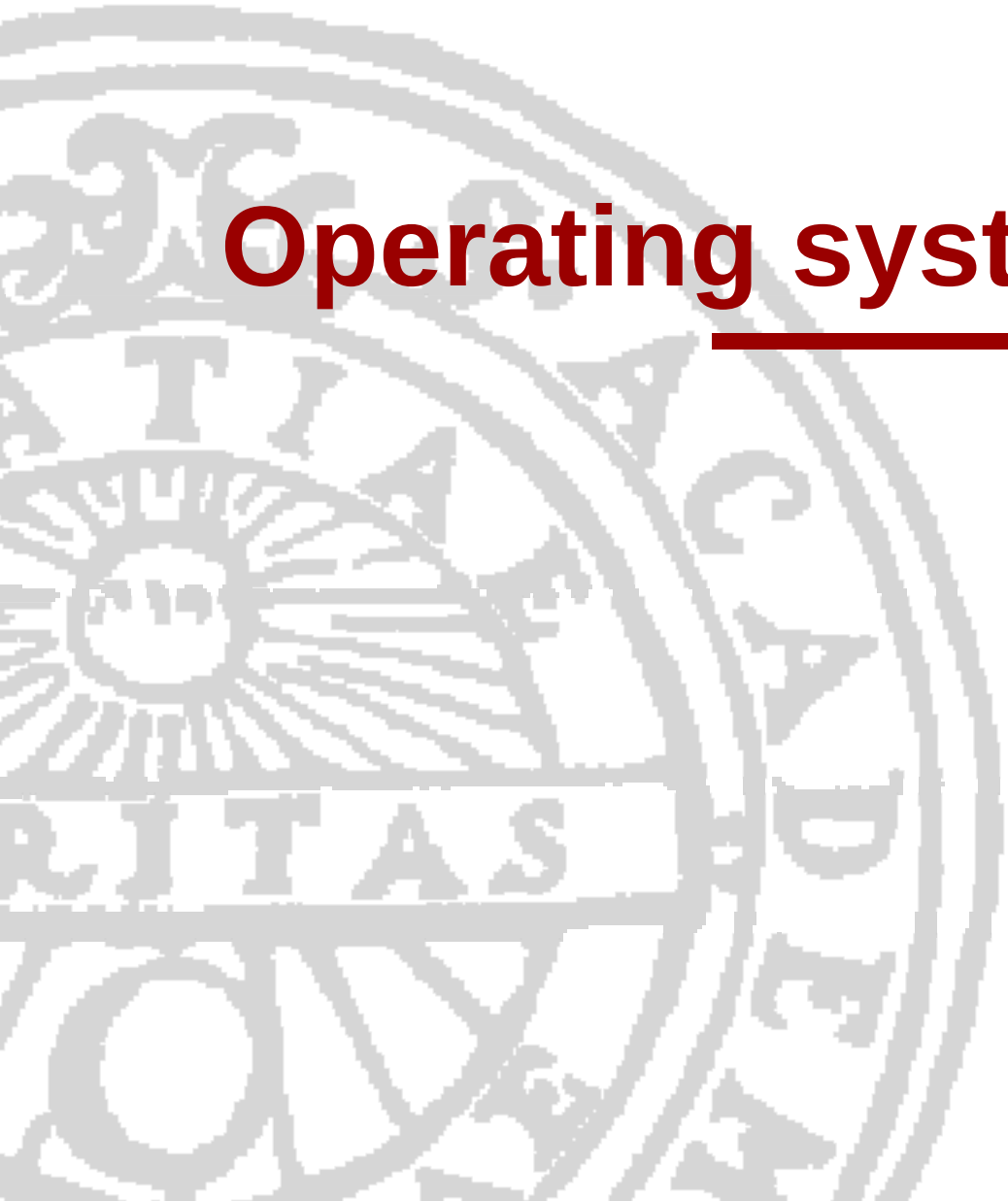
```
int p[10], *ptr;  // Both p and ptr are pointers
                  // i.e. can hold addresses.
                  // p is already pointing to a
                  // fixed location and cannot
                  // be changed. ptr is still
                  // to be initialized.
```

p[i] is an int value.
p, &p[i] and (p+i) are addresses or pointers.
*p is the same as p[0] (They are both int values)
*(p+i) is the same as p[i] (They are both int values)

Pointer arithmetic

```
ptr = p;  // or ptr = &p[0]
ptr +=2;
     //  ptr = ptr + 2 * sizeof(int) = ptr+8 bytes
     //  ptr = 3000 + 8 = 3008 => ptr = &(p[2]);
```

ERROR: `p = ptr;` because “p” is a constant address, points to the beginning of a static array.



Operating system overview

UPPSALA UNIVERSITET

Informationsteknologi

Operating System

- A program that controls the execution of application programs
- An interface between applications and hardware

UPPSALA UNIVERSITET

Informationsteknologi

Operating System Objectives

- Convenience
  - Makes the computer more convenient to use
- Efficiency
  - Allows computer system resources to be used in an efficient manner
- Ability to evolve
  - Permit effective development, testing, and introduction of new system functions without interfering with service

Layers of Computer System

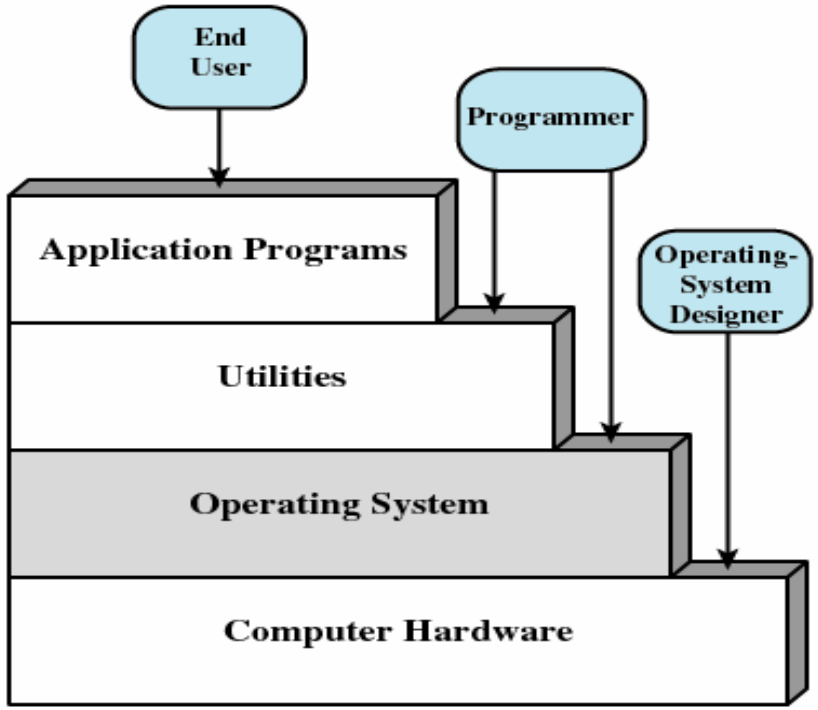


Figure 2.1 Layers and Views of a Computer System

UPPSALA UNIVERSITET

Informationsteknologi

Services Provided by the Operating System

- Program development
  - Editors and debuggers
- Program execution
- Access to I/O devices
- Controlled access to files
- System access

UPPSALA UNIVERSITET

Informationsteknologi

Services Provided by the Operating System

- Error detection and response
  - Internal and external hardware errors
    - Memory error
    - Device failure
  - Software errors
    - Arithmetic overflow
    - Access forbidden memory locations
  - Operating system cannot grant request of application

Services Provided by the Operating System

- Accounting
  - Collect usage statistics
  - Monitor performance
  - Used to anticipate future enhancements
  - Used for billing purposes

Operating System

- Responsible for managing resources
- Functions same way as ordinary computer software
  - It is a program that is executed
- Operating system relinquishes control of the processor for other software to run and depends on the processor to allow it to regain control

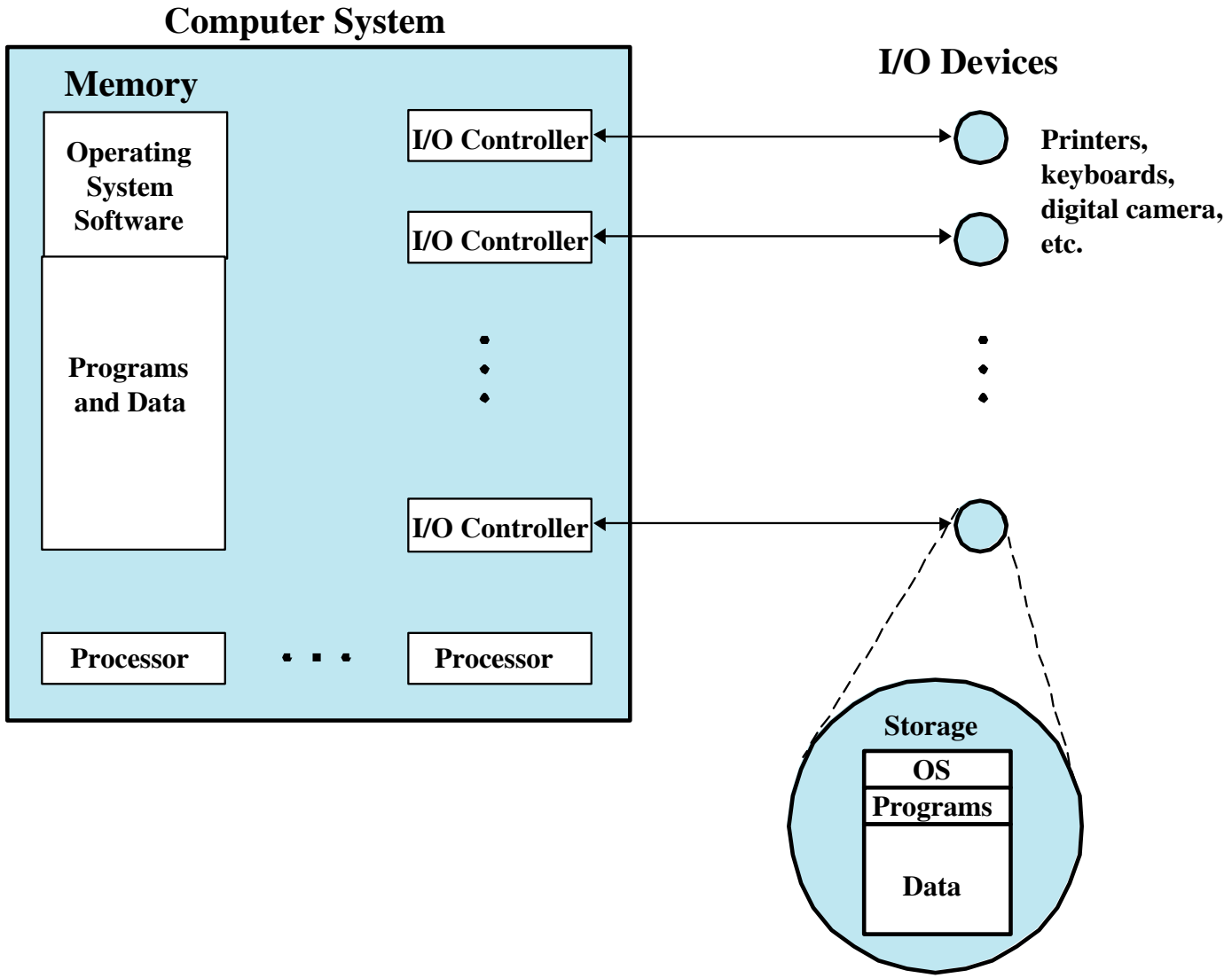


Figure 2.2 The Operating System as Resource Manager

UPPSALA UNIVERSITET

Informationsteknologi

Kernel

- Portion of operating system that is in main memory
- Contains most frequently used functions
- Also called the nucleus

Evolution of an Operating System

- Hardware upgrades plus new types of hardware
- New services
- Fixes

Memory Protection

- User program executes in user mode
  - Certain instructions may not be executed
- Monitor executes in system, or kernel, mode
  - Privileged instructions are executed
  - Protected areas of memory may be accessed

I/O Devices Slow

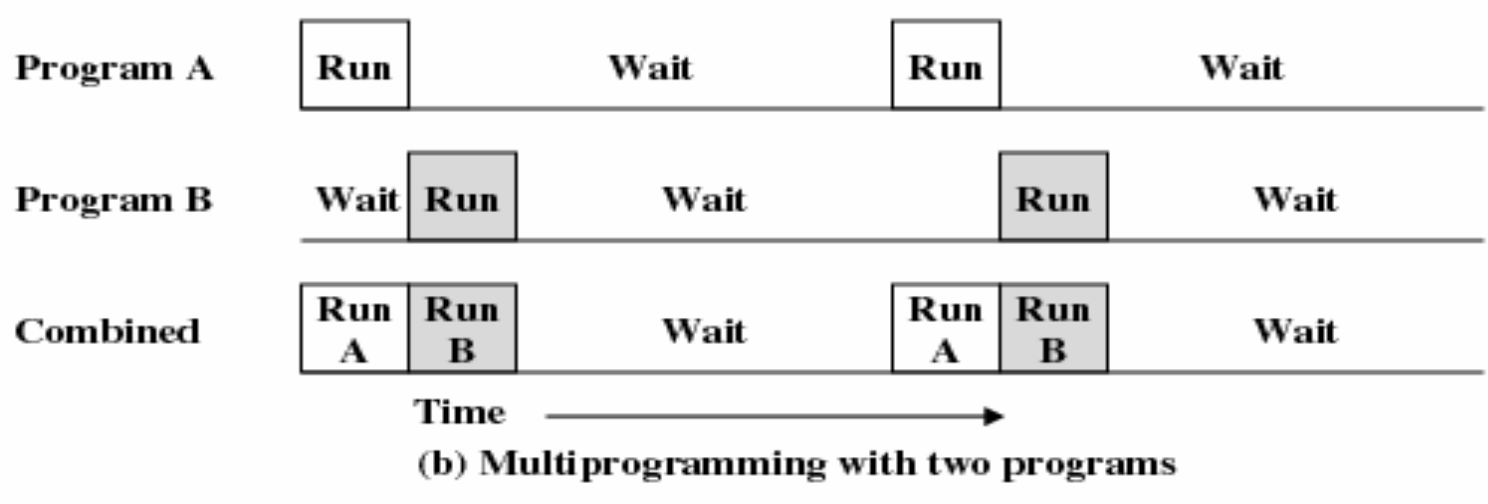
Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

Percent CPU Utilization $= \frac{1}{31} = 0.032 = 3.2\%$

Figure 2.4 System Utilization Example

Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job



(b) Multiprogramming with two programs

Informationsteknologi

Time Sharing

- Using multiprogramming to handle multiple interactive jobs
- Processor's time is shared among multiple users
- Multiple users simultaneously access the system through terminals