Input/Output – 2
I/O Software

CS 342 – Operating Systems

Ibrahim Korpeoglu
Bilkent University
Department of Computer Engineering

Goals of I/O Software

- Device Independence
  - It should be possible to write programs that can access any I/O device without having to specify the device in advance and without knowing the internal details of the device.
    - A program reading a file should be able to read that file
      - From a floppy drive
      - From a hard disk
      - From a tape drive
      - From a CD-ROM
    - The programmer should not remodify its program for each device

Goals of I/O Software

- Uniform Naming
  - Access to all files and devices should be done by a uniform naming scheme.
    - /usr/home/ali/project.c is a pathname and filename that specifies a file which can be read and written.
    - /mnt/floppy is a pathname that specifies the floppy drive which can be read and written.
  - Therefore pathnames are uniform naming schemes that can be used to refer to any device or file.

Goals of I/O Software

- Error handling
  - Errors need to be handled at the lower layer possible
    - At the controller if possible
    - At the device drivers of possible
    - Then at the application
  - Example: reading a disk block
    - Controller computes the checksum. Detect the errors. Tries to correct them if it can.
    - If controller can not correct a block error, device driver requests the disk controller to retrieve the same block again, hoping that this time the block will not be erroneous.

Goals of I/O Software

- Synchronous versus Asynchronous Transfers
  - I/O hardware operates in an asynchronous manner.
    - CPU issues a request to read a block.
    - Disk controller retries the block and then interrupts the CPU
    - Asynchronous I/O is interrupt driven.
  - Programs are easily written of they use synchronous I/O
    - A read() function in a program will block until the data is available.
    - Application programmers can not easily program using interrupts.
    - Synchronous I/O is blocking I/O
  - Operating System should implement synchronous I/O model for applications using the underlying asynchronous I/O model of the hardware.

Goals of I/O Software

- Buffering
  - Operating System should buffer data
    - The speed at which application reads or writes data can not always match the speed at which the data is transferred to or from devices
- Sharing or Devices
  - Some devices are sharable by many users at the same time: hard disk.
  - Some devices are dedicates to a single user while he/she is using the device: tape, printer, etc.
  - OS should manage these devices and deal with deadlocks that are the result of using dedicated devices.

Programming Input and Output

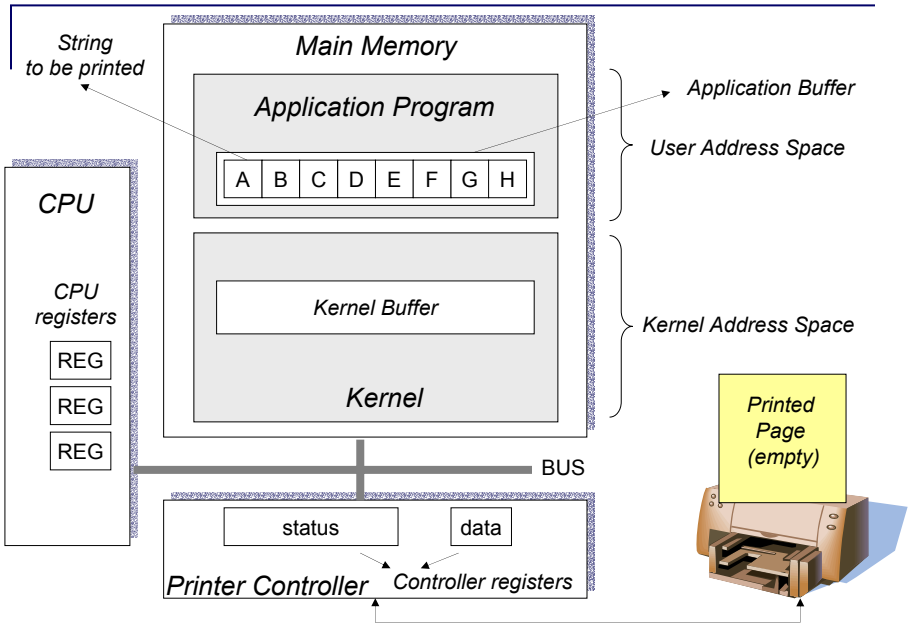
- Three methods
  - Programmed I/O
  - Interrupt driven I/O
  - I/O using DMA

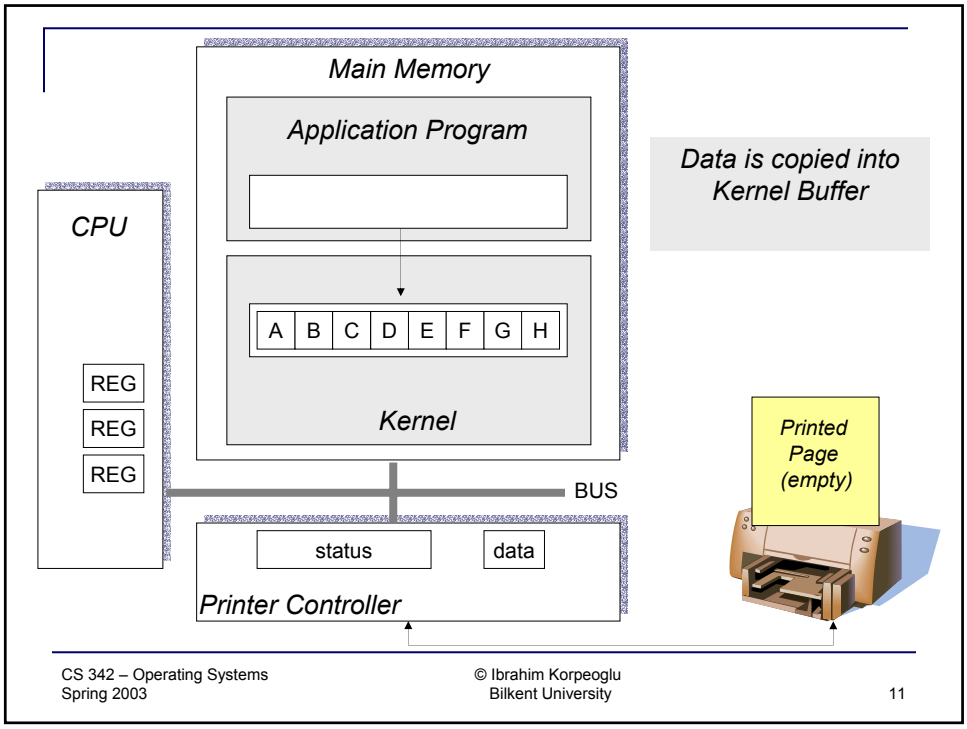
Programmed I/O

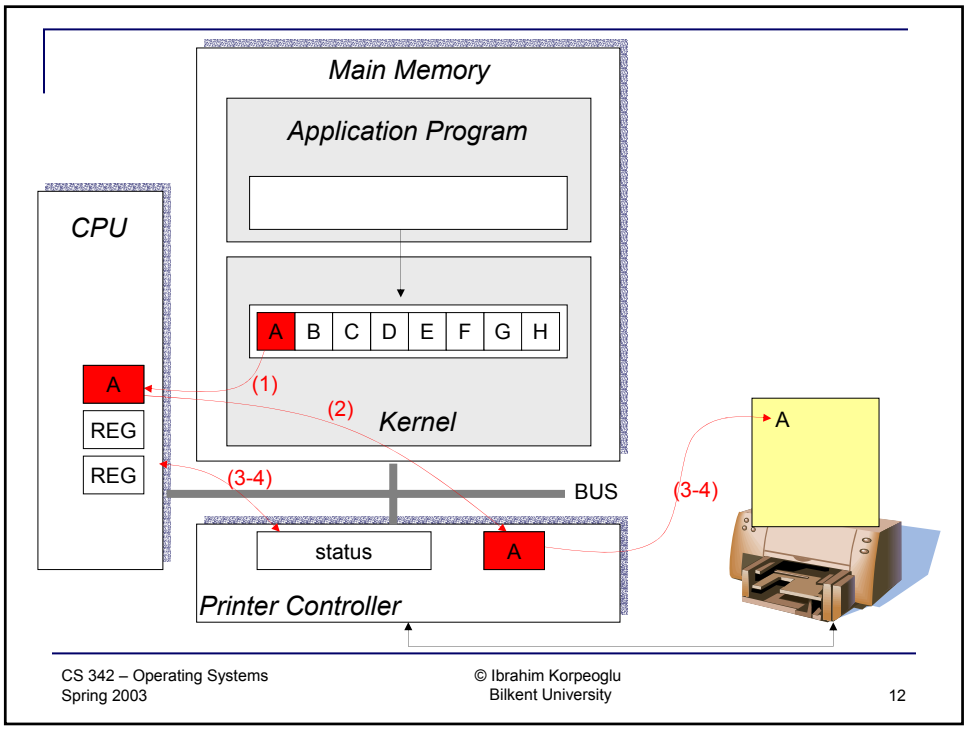
- In programmed I/O, CPU is always busy with I/O until I/O gets completed.
  - Fine for single process systems
    - MS-DOS
    - Embedded systems
  - But not a good approached for multi-programing and time-sharing systems

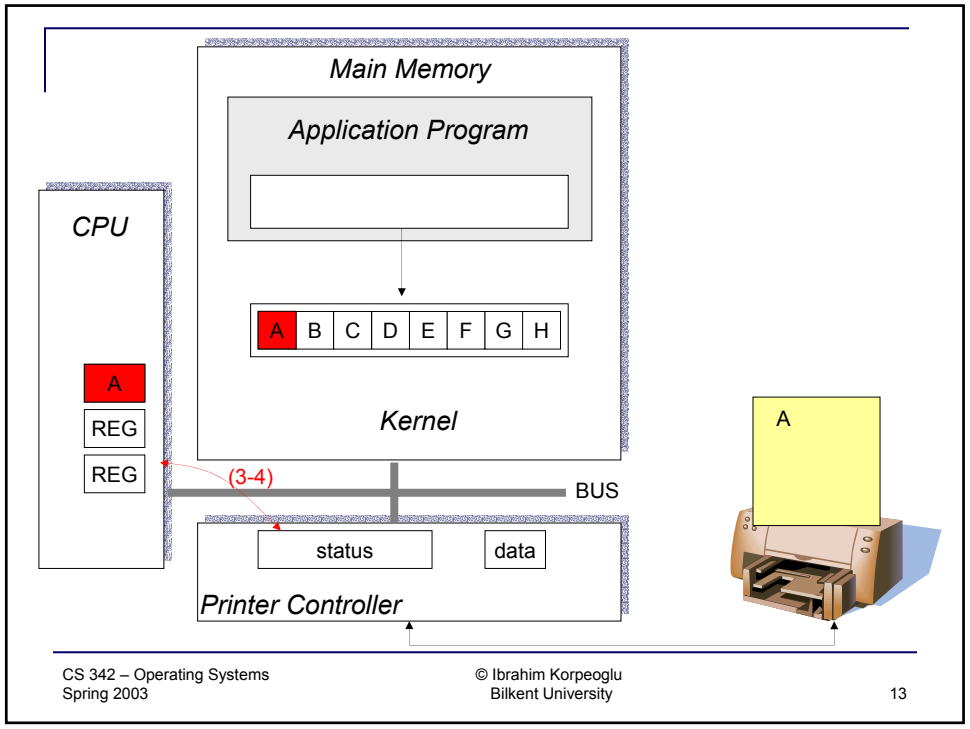
Programmed I/O - Example

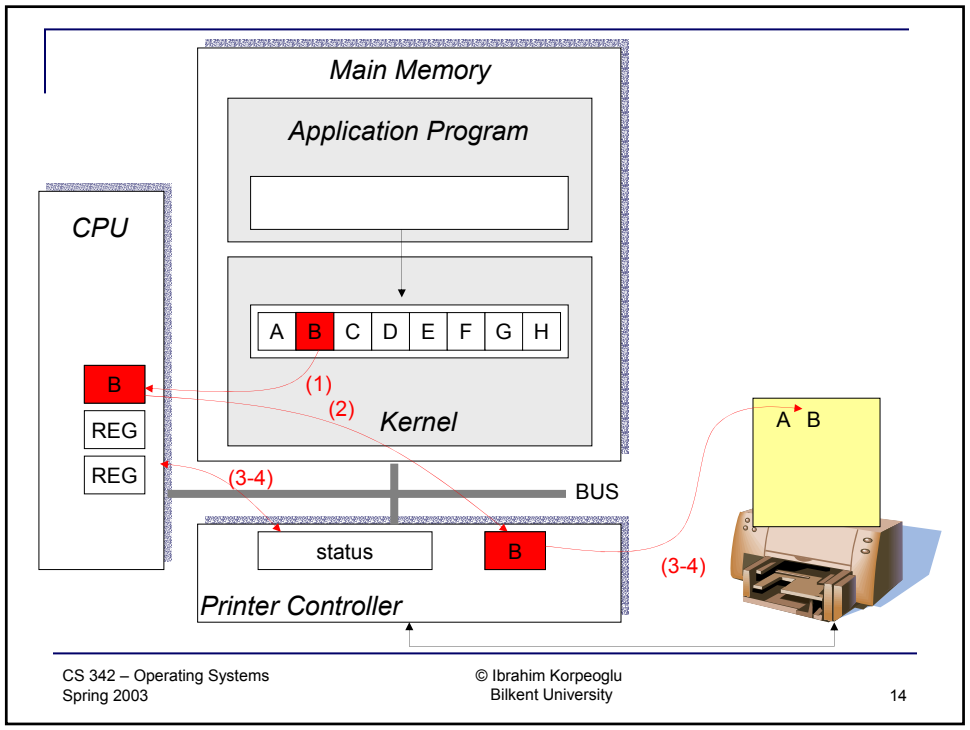
- Assume a program that wants to print a string to a printer.
- Assume string is “ABCDEFGH”
  - 8 character long string
- Assume printer has 1 byte of data buffer to put the character that needs to be printed.

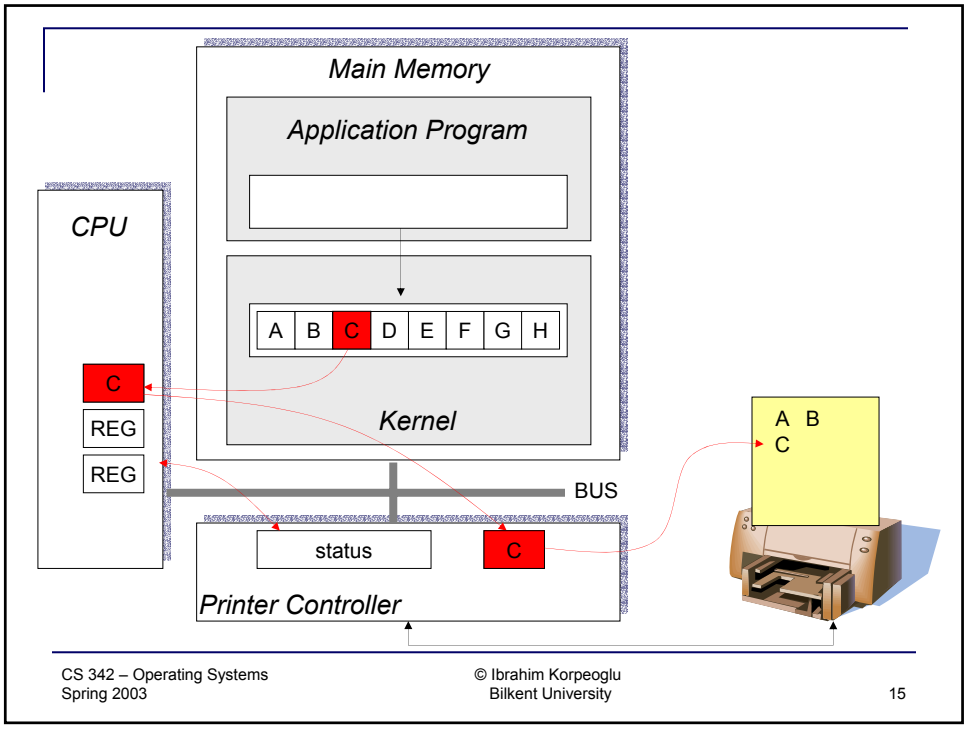


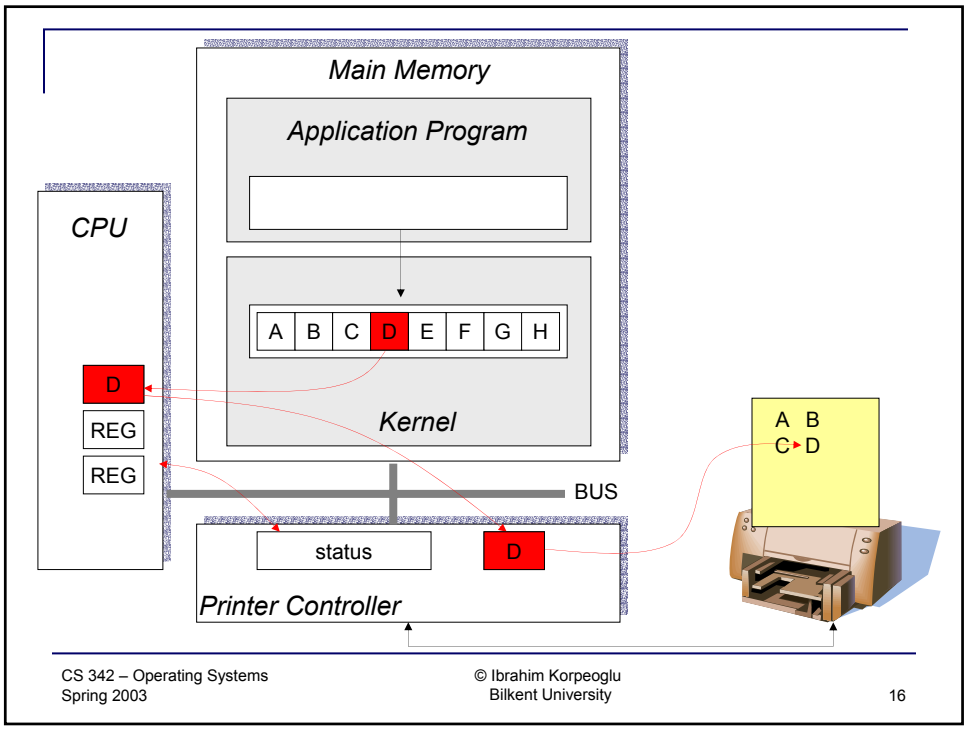

Main Memory
Application Program
A B C D E F G H
Kernel
CPU
REG
REG
REG
BUS
status
data
Printer Controller
Data is copied into Kernel Buffer
Printed Page (empty)

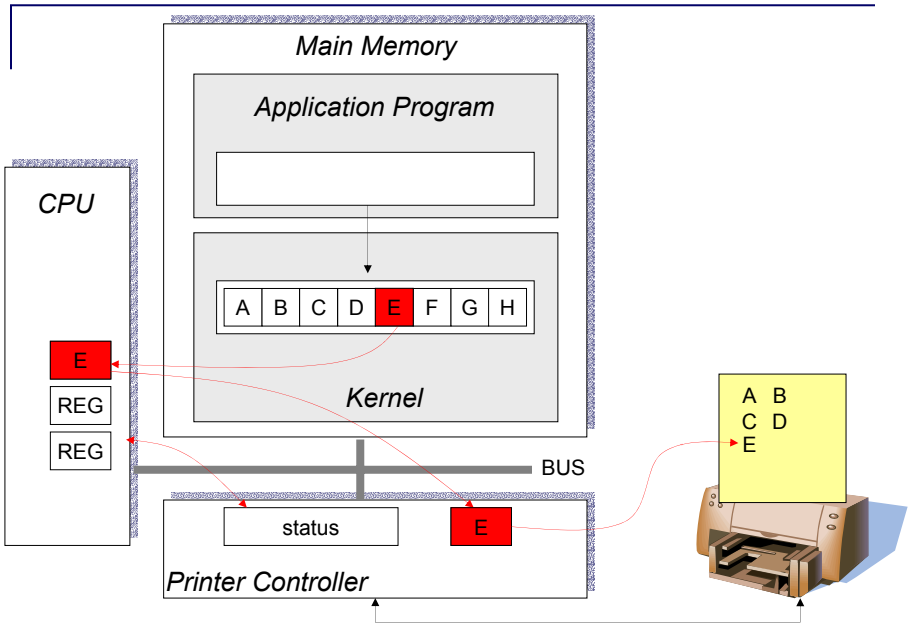

Main Memory
Application Program
A B C D E F G H
Kernel
CPU
A
REG
REG
(1)
(2)
(3-4)
BUS
status
A
Printer Controller
(3-4)
A


Main Memory
Application Program
A B C D E F G H
Kernel
CPU
A
REG
REG
(3-4)
BUS
status
data
Printer Controller
A


Main Memory
Application Program
A B C D E F G H
Kernel
CPU
B
REG
REG
(1)
(2)
(3-4)
BUS
status
B
Printer Controller
A B
(3-4)


Main Memory
Application Program
A B C D E F G H
Kernel
CPU
C
REG
REG
BUS
status
C
Printer Controller
A B
C


Main Memory
Application Program
A B C D E F G H
Kernel
CPU
D
REG
REG
BUS
status
D
Printer Controller
A B
C D



How would we implement copy in kernel!

```
/*
   buffer is user buffer
   p is kernel buffer
   count is number of bytes to be copied
*/
copy_from_user(buffer, p, count)
{
        for (i=0; i < count; ++i) /* loop in every character */
        {
            while (*printer_status_register != READY)
            {};   /*busy waiting – loop until ready*/
            *printer_data_register = p[i];  /* output one
                                               character */
        }
        return_to_user();
}
```