# Blocking and Non-blocking IO
# Software Technologies - Lecture 7

Saurabh Barjatiya

International Institute Of Information Technology, Hyderabad

2 March, 2010

# Contents

1. Blocking I/O
   - Introduction
   - poll
   - select
2. Non-Blocking I/O
   - Introduction
3. Asynchronous I/O
   - Introduction

# Introduction

- By default read() / write() functions are blocking in nature, that is if there is no input and EOF is not reached then they wait till some input arrives or stream is closed (ie EOF is reached).
- Blocking is not same as buffering, even if one byte of input is available read() function would return. Buffering waits till the buffer is full or some special sequence like '\r\n' or just '\n' is read.

# poll

- We can use poll() when using Blocking I/O to know whether the particular function call will block or not.
- poll() can also be used to check whether the connection is closed (Hangup) or is it still running. For this purpose we have to attempt a read/write on socket before we check whether it is closed or not.

# select

- We can use select() if we are waiting for input on multiple file descriptors.
- We can choose whether we are waiting for reading or writing or exceptions.
- select() and poll() are not only for network, they can be used on normal file descriptors including 0 (stdin), 1 (stdout), 2 (stderr).
- We can specify very precise time to wait for using select() calls.

# Contents

1. Blocking I/O
   - Introduction
   - poll
   - select
2. Non-Blocking I/O
   - Introduction
3. Asynchronous I/O
   - Introduction

# Introduction

- Using fcntl() we can change the file descriptor options so that read() and write() on it do not block.
- In such case -1 is returned if there is no data too or if error occurs and 0 if EOF is reached.
- To differentiate between error and no data, read ‘errno’ variable. If its value is EWOULDAGAIN, then it means -1 is returned to indicate no data, else some error occurred.
- Setting O_NONBLOCK affects accept() and connect() functions calls too.

# Contents

1. Blocking I/O
   - Introduction
   - poll
   - select
2. Non-Blocking I/O
   - Introduction
3. Asynchronous I/O
   - Introduction

# Introduction

We can also perform I/O asynchronously. This requires use of signals. We have to enable asynchronous I/O (O_ASYNC) on file descriptor using fcntl(). We also have to use fctnl() to make current process owner of file descriptor, so that signals are delievered to current process (F_SETOWN) .

We have to use signal handling for caching signals SIGIO and SIGHUP. We also have to use ioctl() to enable asynchronous I/O and signals on file descriptor.

The program which uses above process is very efficient as it does network/file I/O when it can and at other times it can do other processing. Also all this is done without polling and without using multi-threading or multi-processing.