Implementation of a Single Threaded User level Asynchronous I/O Library using the Reactor pattern

Ramakrishnan Kalicut

Master's Thesis Defense

Jan 20, 2003

Committee:

Dr. Jerry James (Chair)

Dr. Gary Minden

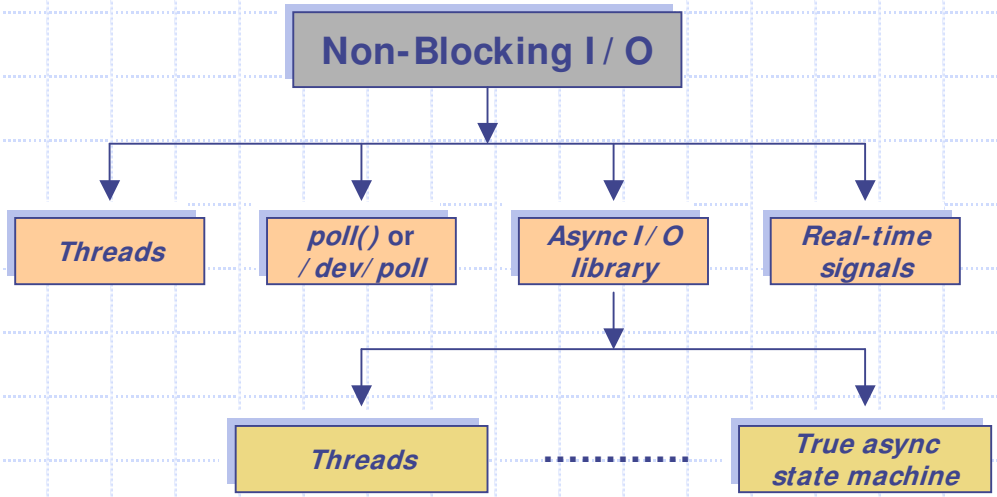
Dr. Arvin Agah

Overview

- Introduction
- Motivation
- Related Work
- Single-threaded asynchronous I/O library
- Performance Evaluation
- Conclusions & Future work

Introduction

- Due to increase in data access
  - Conventional I/O has become a bottleneck
    - Scientific applications involve huge amounts of I/O. I/O device speed is 2-3 orders of magnitude less than the CPU
  - Need to mask disk latencies with application processing



Motivation

- Disadvantages of multi-threaded asynchronous I/O
  - Thread maintenance & context switching overhead
  - Poor scalability
  - Complex synchronization requirements
- Solution
  - Single-threaded library on top of an event-driven framework

Related Work

- Asynchronous I/O implementations
  - Entire subsystem in an asynchronous model
    - Built from scratch
    - Microsoft Windows NT I/O subsystem
  - Separate asynchronous interface
    - Interface built coexists with existing synchronous interfaces
    - Most Linux implementations

Related Work

- Asynchronous I/O implementation models
  - Multi-threading
    - One thread per process – *glibc/librt* version
    - Using a thread pool – Solaris *laio* version
  - Hybrid approach – multi-threading and use of the asynchronous behavior of underlying hardware.
    - Need specific routines at the device-driver level
    - SGI KAIO implementation
  - True asynchronous state machine
    - Sequence of non-blocking steps, with state transitions driven by IRQ techniques and event threads
    - AIO interface being developed at IBM

Single-threaded AIO Library

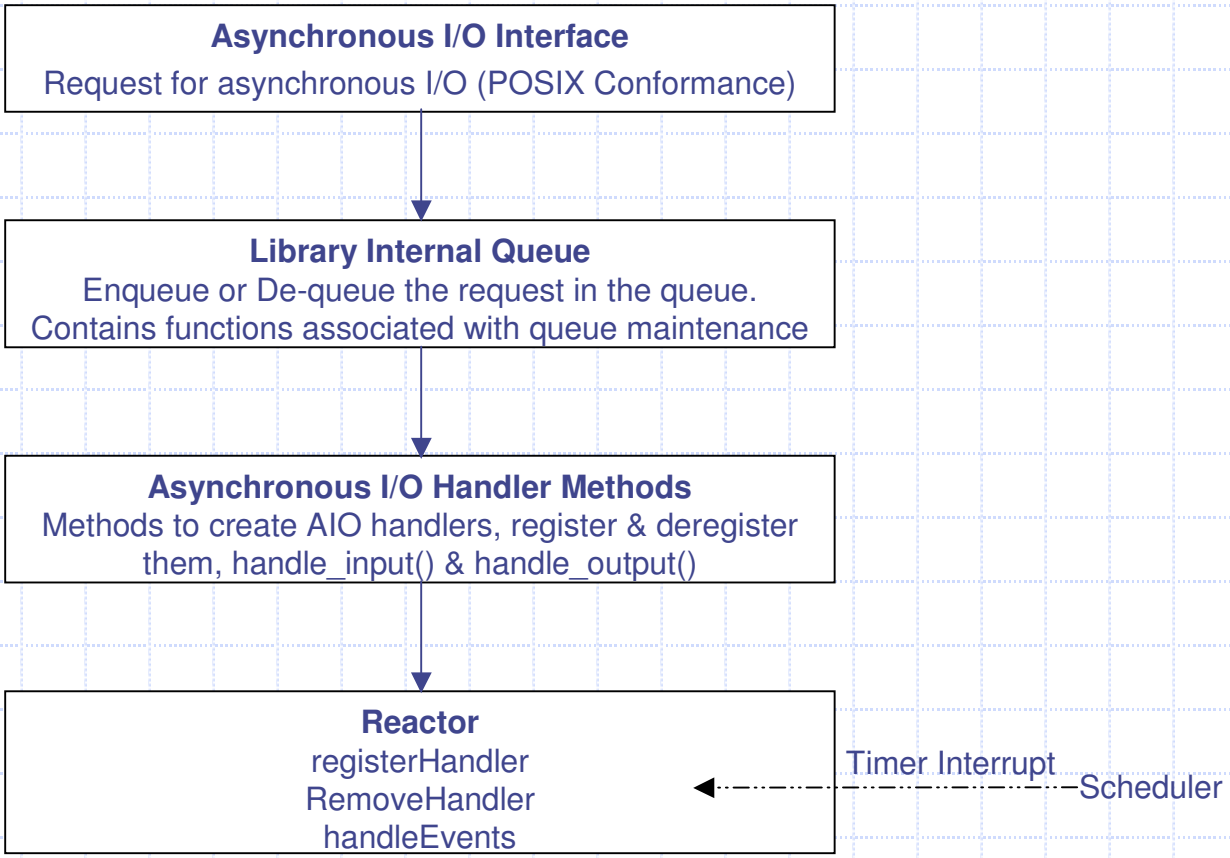
- A POSIX compliant asynchronous I/O interface
  - Created on top of Reactor, an object-oriented event driven framework
- Library Components
  - The interface
  - Library internal queue
    - *free list*
    - *run list*
    - *done list*
  - Reactor
    - *select()* at its core
    - Time triggered

Single-threaded AIO Library

- Library Initialization/Finalization
  - Allocate resources for some requests in *free list*
    - Defers later allocations for each request
  - Set the timer to schedule the Reactor
  - Resources freed as part of finalization

Single-threaded AIO Library

- Interaction among various components



Single-threaded AIO Library

- Data structures available to the user

Asynchronous Initialization Block

```
struct aioinit{
   int aio_threads;
   int aio_num;
   int aio_locks;
   int aio_usedba;
   int aio_debug;
   int aio_numusers;
   int aio_reserved[2];
};
```

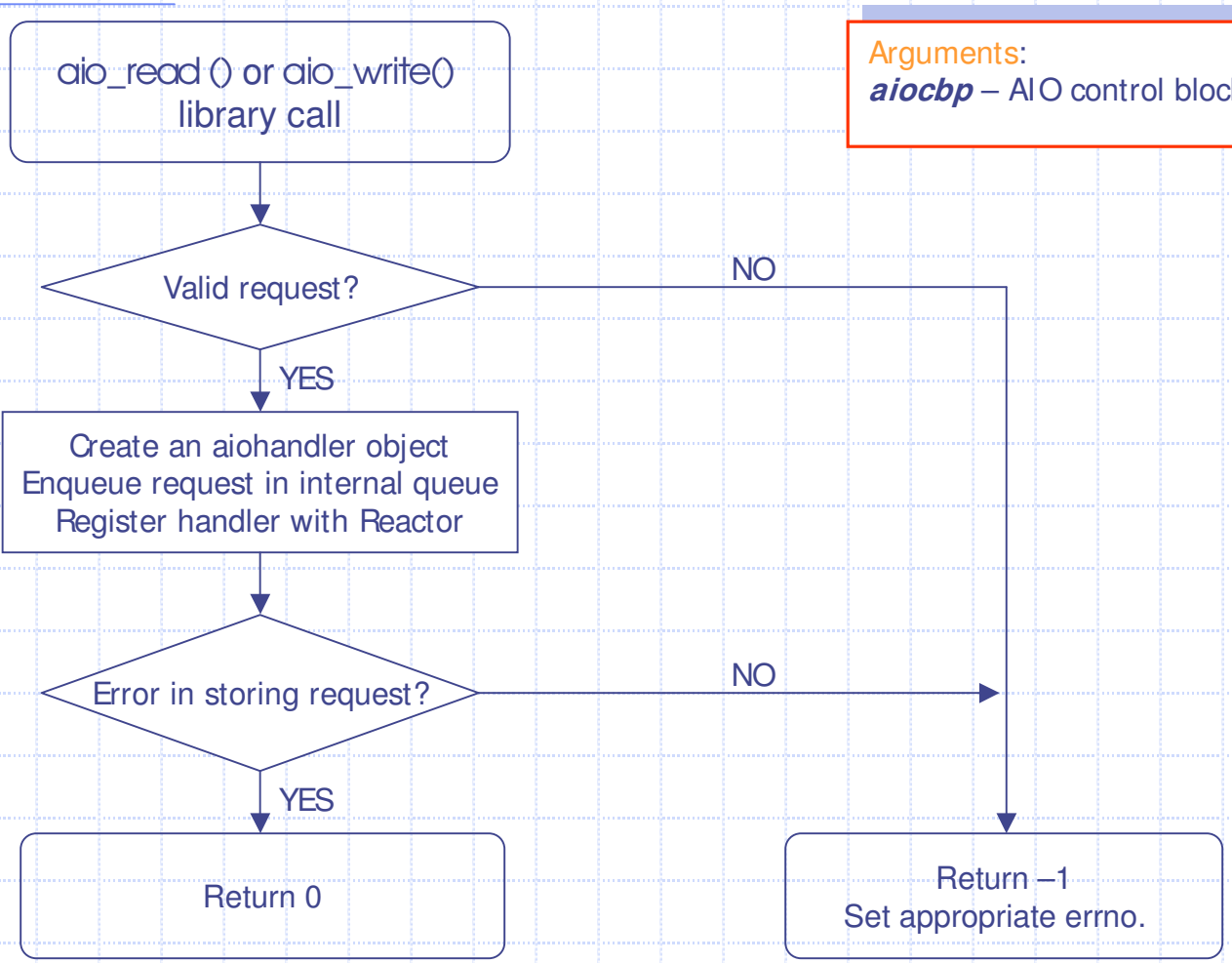
Asynchronous I/O Control Block

```
struct aiocb{
   int aio_fildes;
   int aio_lio_opcode;
   int aio_reqprio;
   volatile void *aio_buf;
   size_t aio_nbytes;
   struct sigevent aio_sigevent;
   off_t aio_offset;
};
```

Single-threaded AIO Library

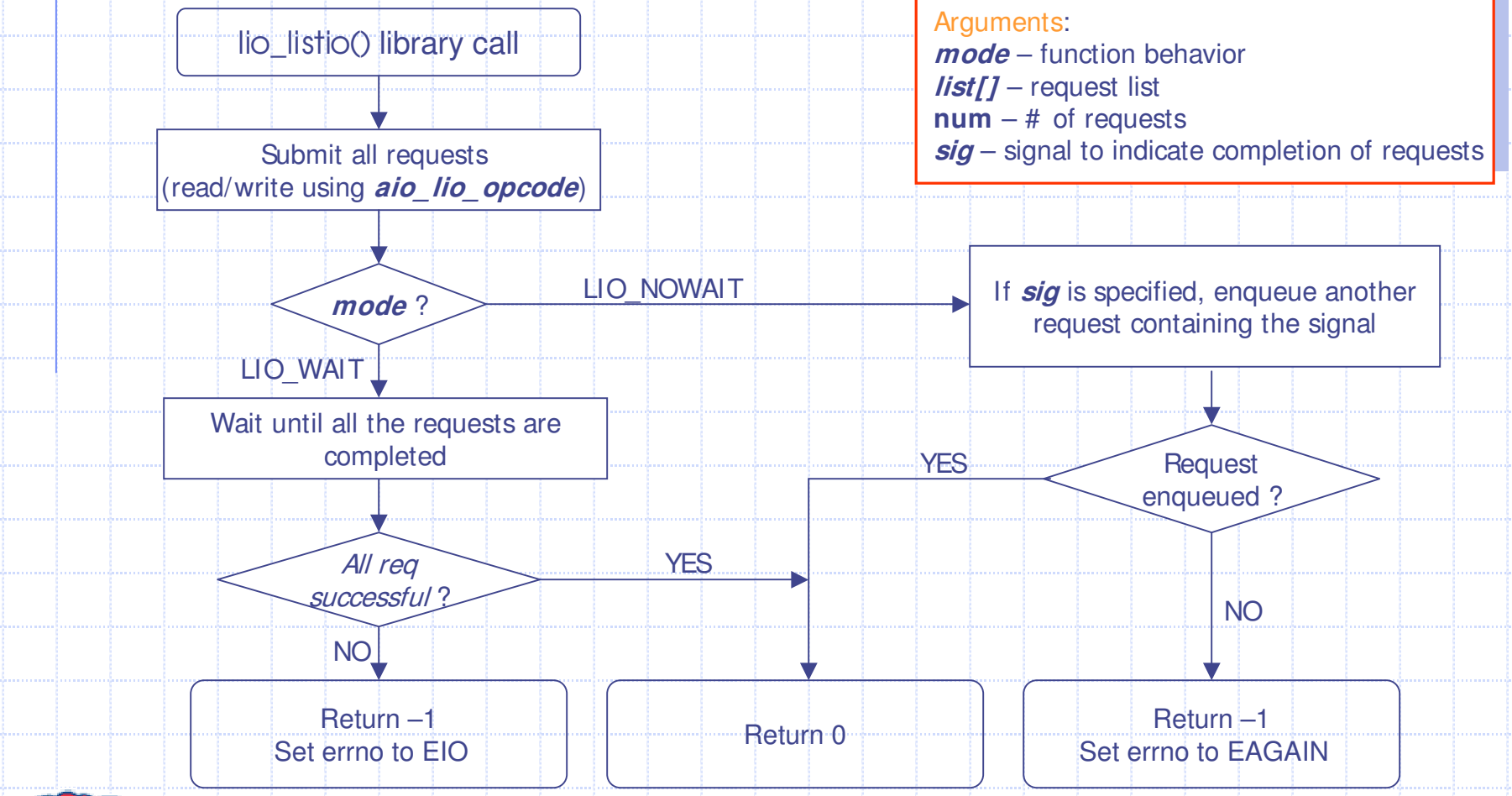
- Library Features
  - Initialization primitives – *aio_init()*
    - Allocate resources
    - Set timer to schedule the Reactor
  - Single asynchronous read/write request
  - Batch asynchronous read/write requests
  - Cancellation of one or more request
  - Synchronization primitives
  - Status check primitives

Single asynchronous read/write

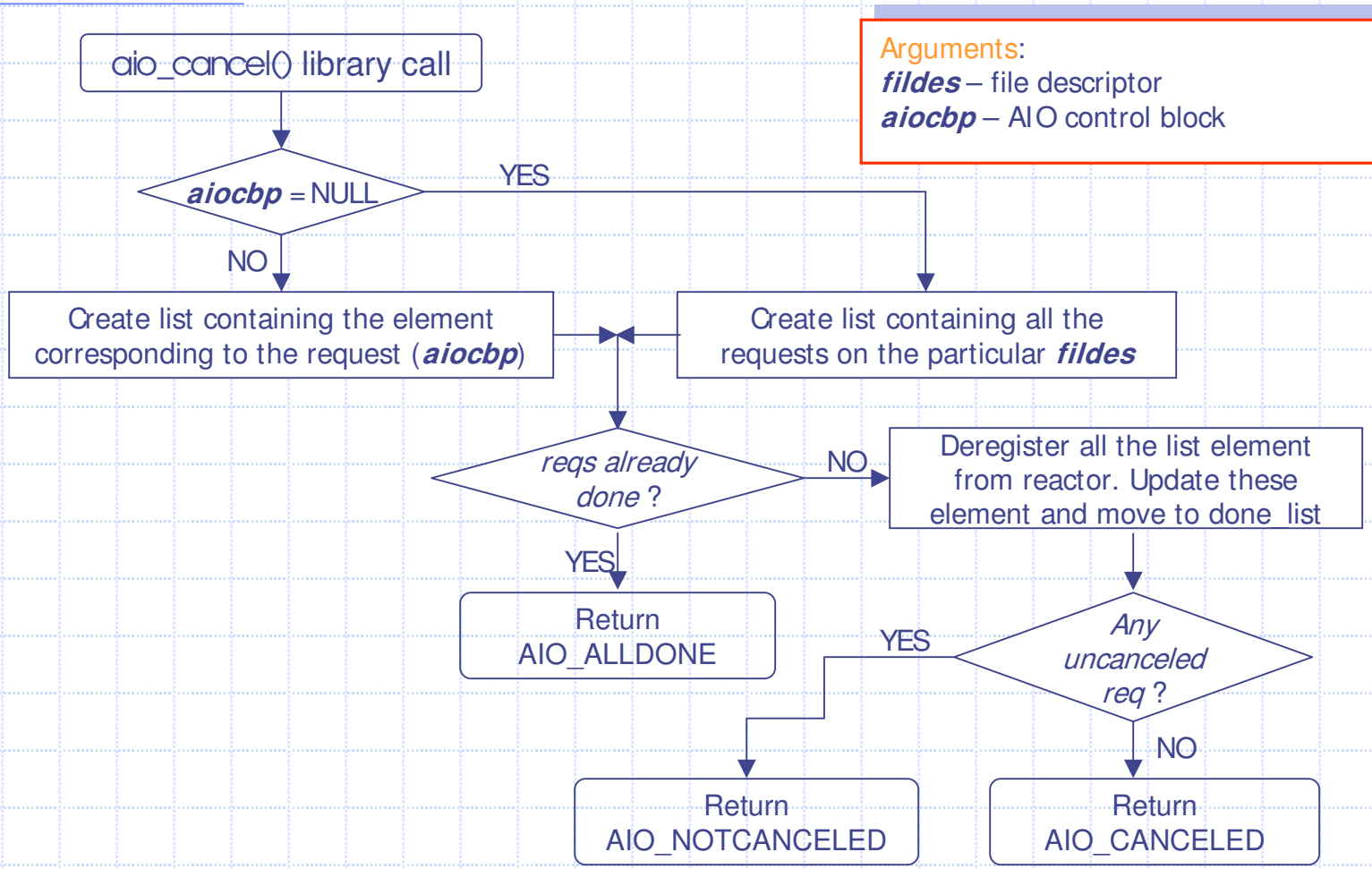


Arguments:
aiocbp – AIO control block

Batch asynchronous read/write



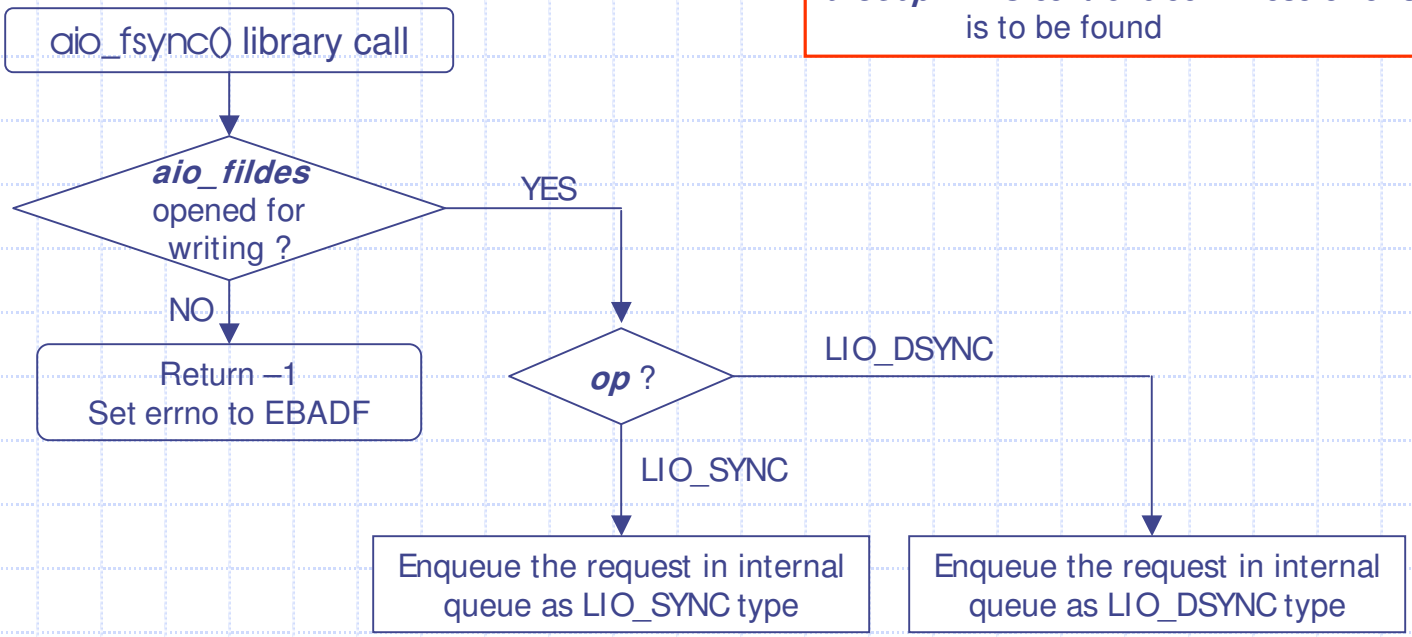
Cancel one or more requests



Synchronization primitives

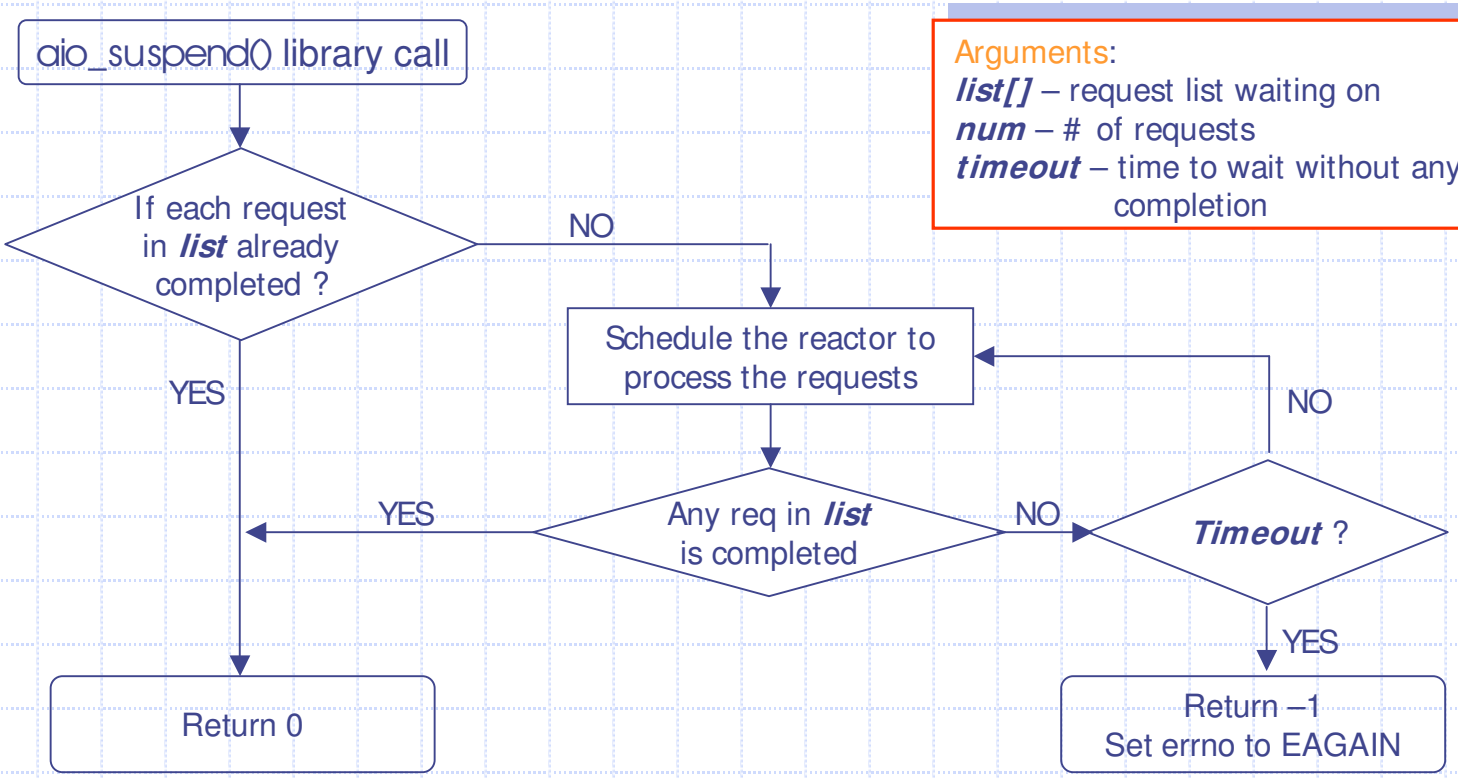
- Make a consistent view of a file

Arguments:
op – fsync or fdatasync flag
aiocbp – AIO control block whose error state is to be found



Synchronization primitives

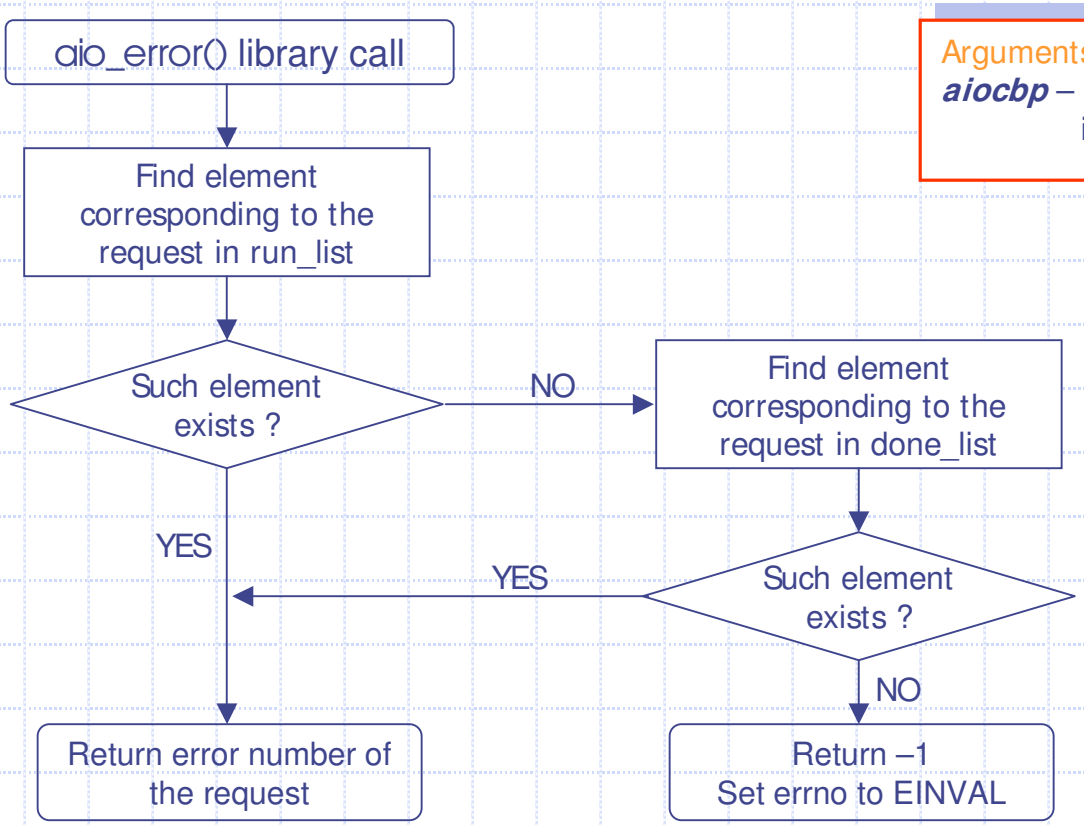
- Wait for one/more requests to complete



Arguments:
list[] – request list waiting on
num – # of requests
timeout – time to wait without any request completion

Status check primitives

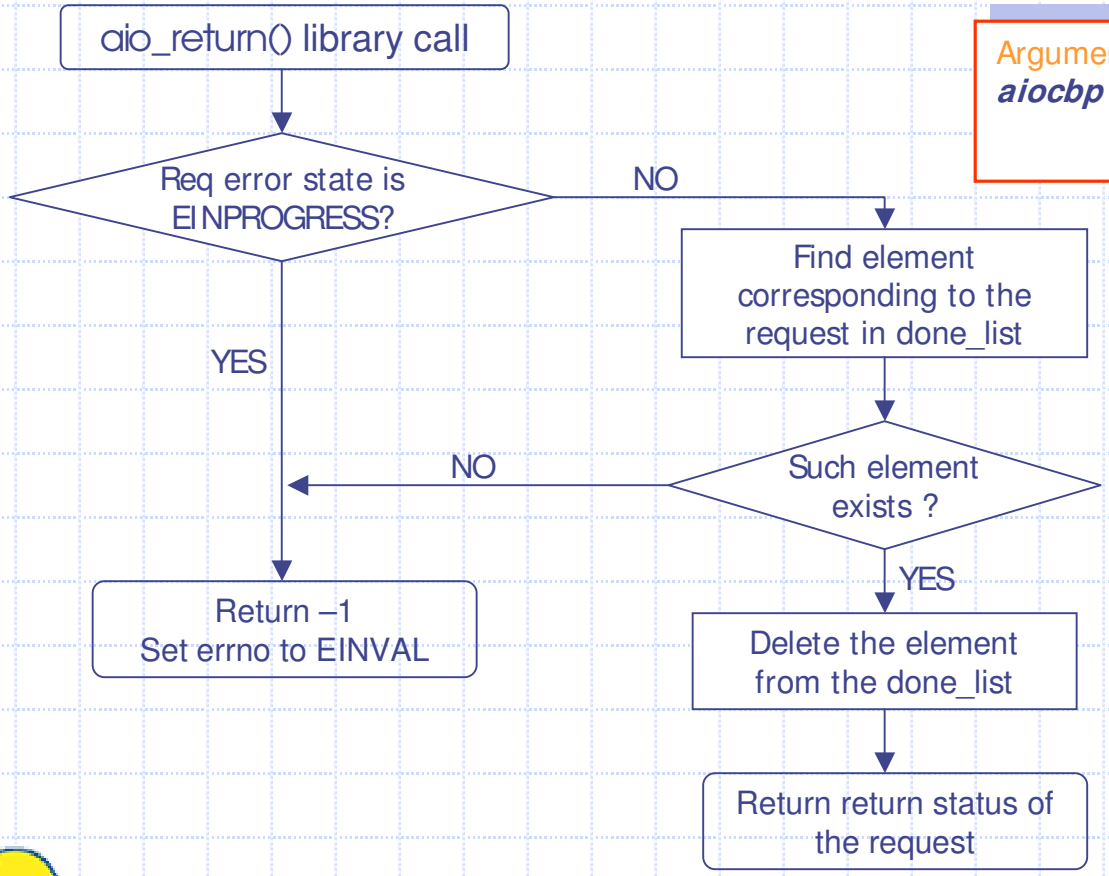
- Check error state associated with a request



Arguments:
aiocbp – AIO control block whose error state is to be found

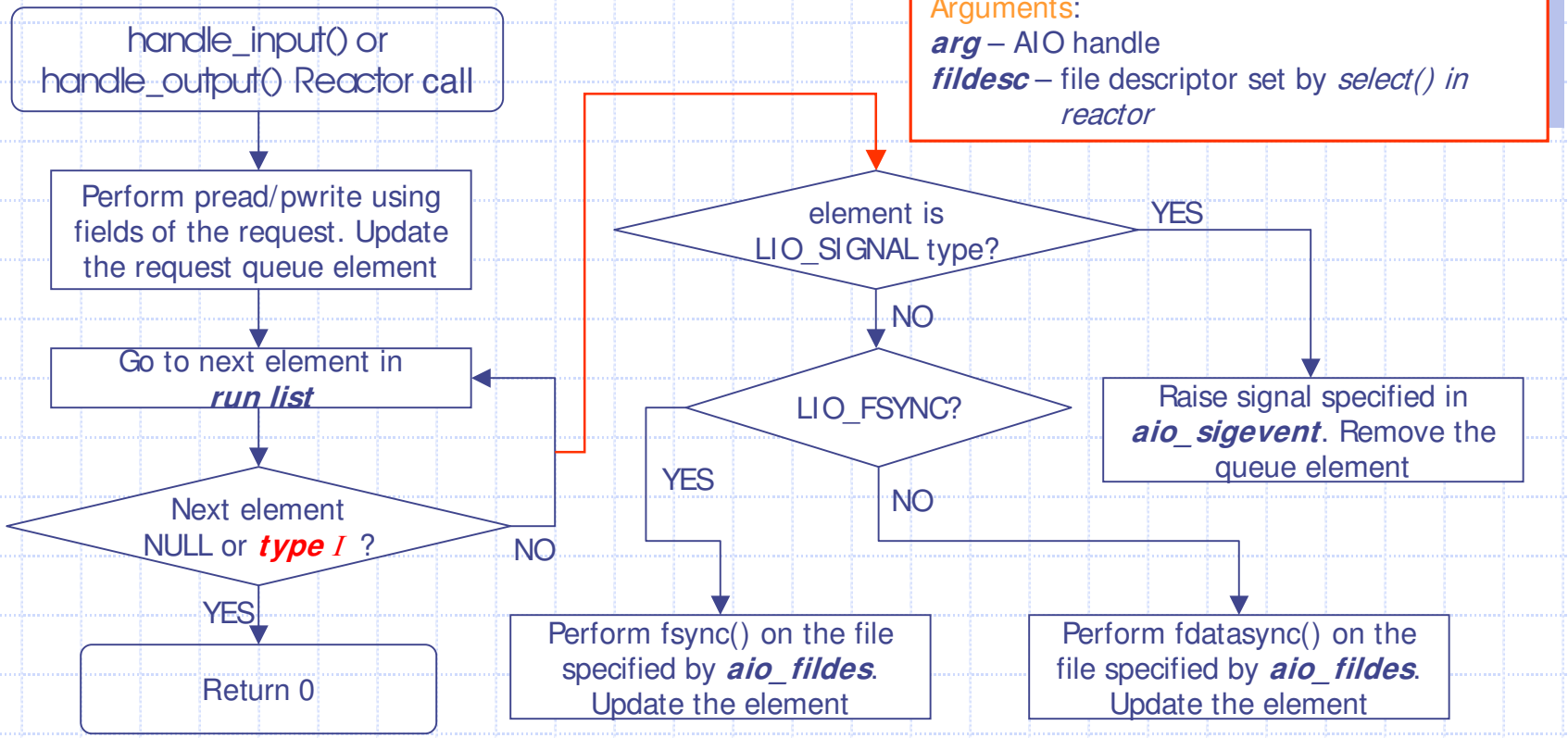
Status check primitives

- Check return status associated with a request



Arguments:
aiocbp – AIO control block whose return status is to be found

Reactor Handle function



Single-threaded AIO Library

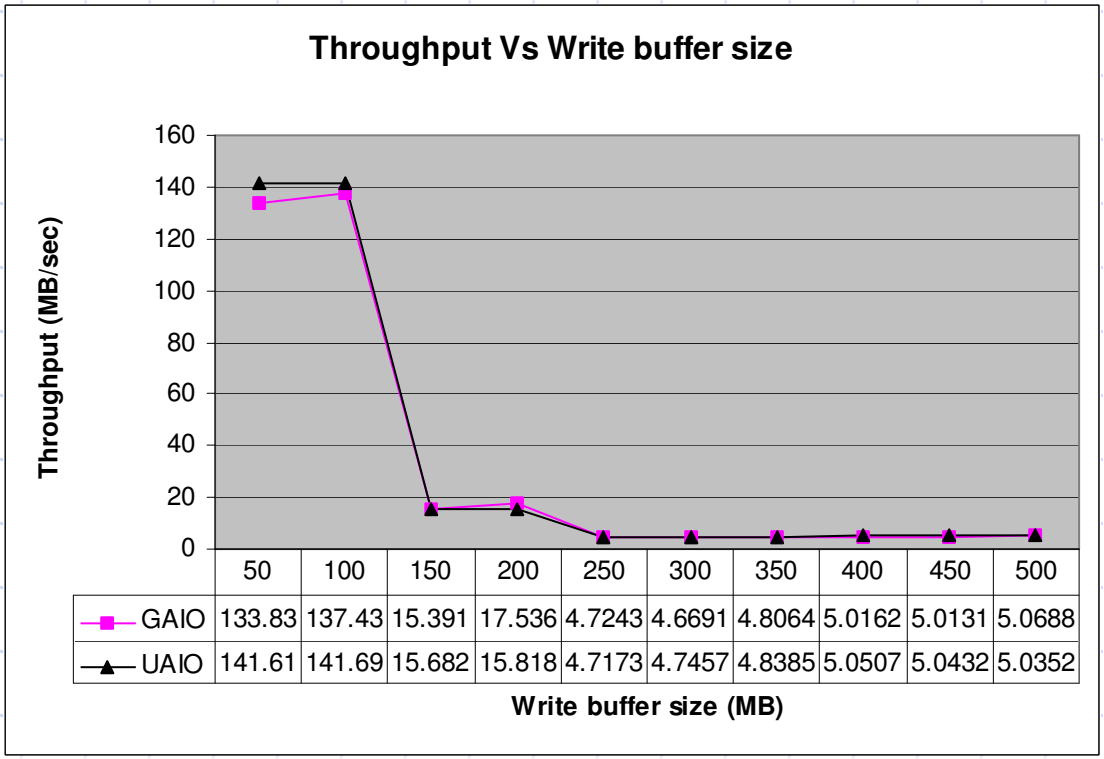
- Limitations
  - As there is only one thread in the process, the user process cannot wait
  - In multi-processor systems
    - UAIO cannot utilize more than one processor
    - The *glibc* version, a multi-threaded implementation, may perform better
      - Each thread can be handled by a separate processor

Performance Evaluation

- Compared with the *glibc* version
  - *glibc* version is multi-threaded in nature
- All tests conducted on a Pentium- III 802.933MHz machine with 256MB RAM
- Tests were performed for file I/O operations
- In the graphs
  - GAIO – *glibc* version
  - UAIO – user-level single-threaded library

Performance Evaluation – Test1

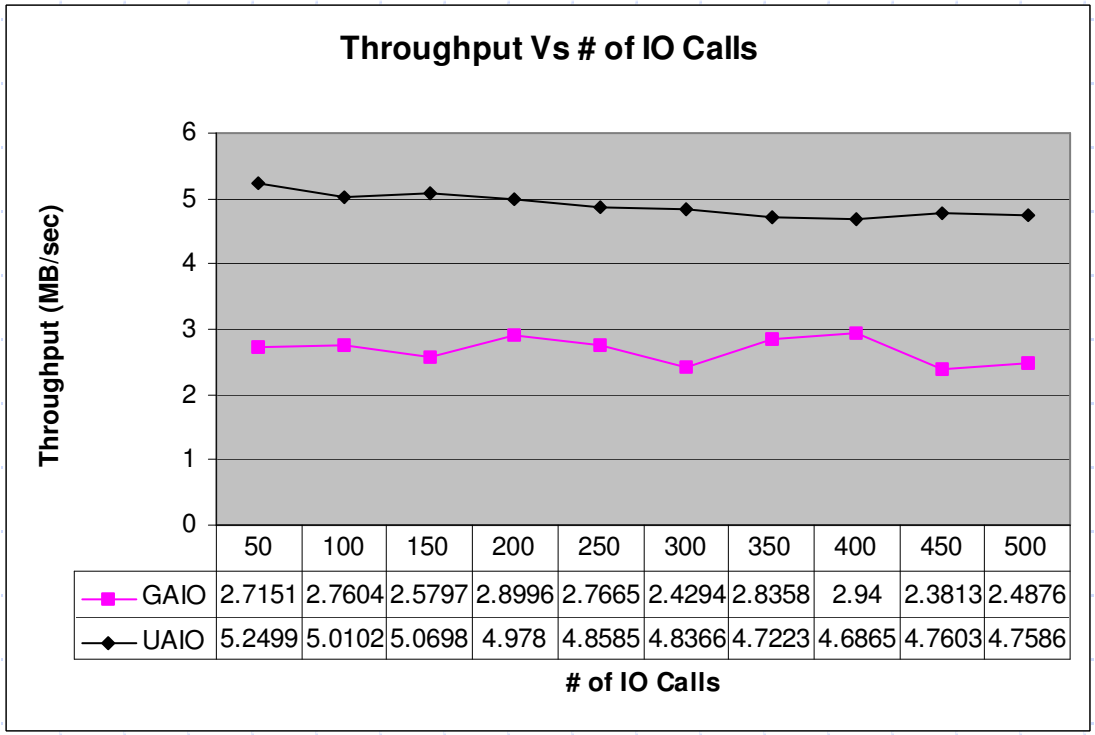
- Buffer size of single write call is varied



	50	100	150	200	250	300	350	400	450	500
GAIO	133.83	137.43	15.391	17.536	4.7243	4.6691	4.8064	5.0162	5.0131	5.0688
UAIO	141.61	141.69	15.682	15.818	4.7173	4.7457	4.8385	5.0507	5.0432	5.0352

Performance Evaluation – Test2

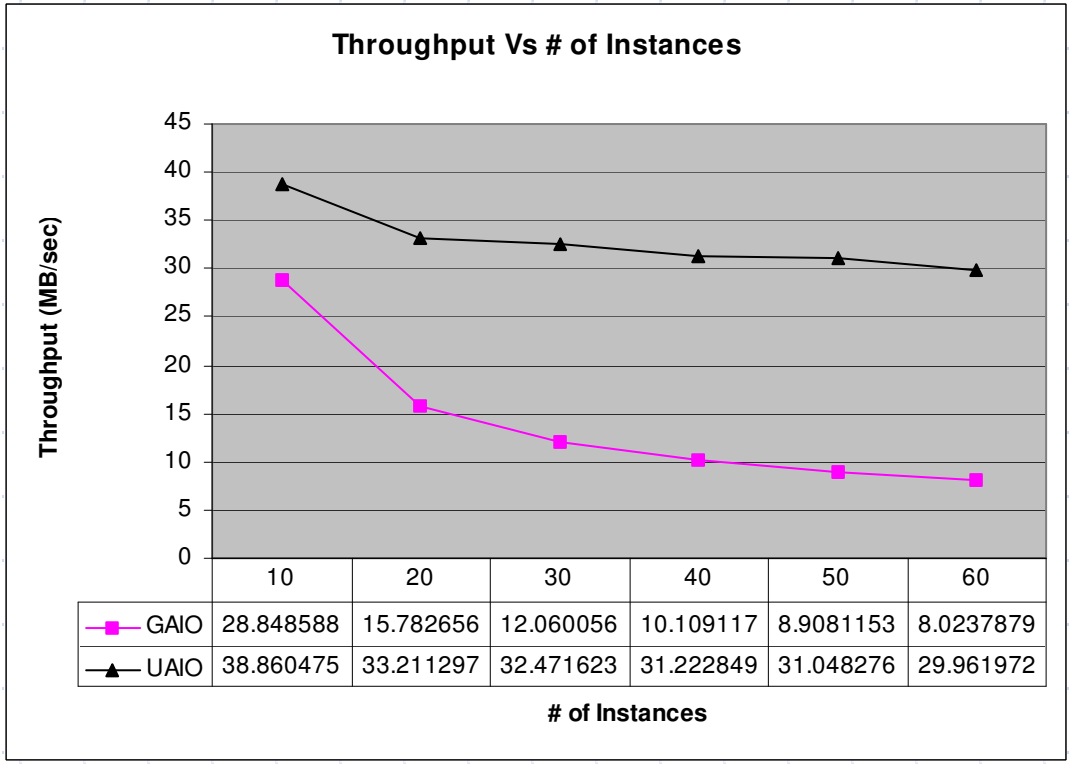
- A load of 300MB is distributed over a number of I/O write calls



	50	100	150	200	250	300	350	400	450	500
GAIO	2.7151	2.7604	2.5797	2.8996	2.7665	2.4294	2.8358	2.94	2.3813	2.4876
UAIO	5.2499	5.0102	5.0698	4.978	4.8585	4.8366	4.7223	4.6865	4.7603	4.7586

Performance Evaluation – Test3

- Number of instances of a process (that does 200 I/O reads) is varied



Conclusions

- Built an user-level asynchronous I/O library on top of a event-driven framework (Reactor)
- Tested POSIX compliance of the library
- Evaluated performance of the UAIO library with system loads in excess of 300MB
  - Performed better in comparison to *glibc* AIO library (multi-threaded library) on a single CPU machine

Future Work

- Wrappers to blocking system calls
  - *wait(), sigsuspend()*
- Conditional-threading
  - Requests for different files handled by separate threads
  - Useful in a multi-processor machine
- Proactor
  - Pattern that supports the demultiplexing and dispatching of multiple event handlers triggered by completion of asynchronous events
  - Proactor can be built on top of UAIO library

Questions ?