SCC.211 Operating Systems

Session 7

Dr Andrew Scott

a.scott@lancaster.ac.uk

1

Overview

- Summary/ recap of:
  - Topic 5: Multi-process Systems
    - Real-Time Systems and Real-Time Scheduling
  - Topic 6: File-systems
- Brief Introduction to
  - Topic 7: Memory Protection

2

5. Multi-process Systems

Summary/ Recap

3

Recap

- Types of multi-process system
- Difference between dispatchers and schedulers
  - Scheduler makes choice, dispatcher actions that choice
- Schedulers
  - Seen examples and comparison/ common metrics for
    - First Come First Served
    - Round Robin

4

Real-Time Systems

- Real-time process
  - Process that delivers results of processing in a given time-span
- Real-time system
  - System in which correctness of computation depends not only on obtaining the right result, but also upon providing the result on time
- Deadline
  - Deadline represents latest acceptable time for
    - Processing result to be considered correct, or…
    - Presentation of result to be valid

5

Real-Time Processes

- Soft Real-Time
  - Deadlines may be broken
    e.g. High-end, professional multimedia
- Hard Real-Time
  - Deadlines critical
    e.g. Embedded/ Control Systems
    - May require safety conformance certification (proof it will <u>always</u> meet deadlines)

6

Real-Time Scheduling

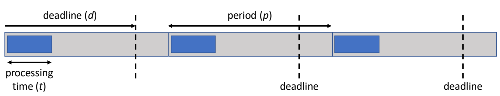
- Driven by events/ deadlines
  - Periodic events …*predictable*
  - Aperiodic events …*unpredictable* -- *beware!*
- Must ensure work can <u>always</u> be scheduled even in <u>worst possible case</u>
  - Never take on too much work …*requires* ***admission-control***
  - Utilisation of resources (including CPU) must never be greater than 100%
    - This must also consider context switching overheads

7

Periodic Tasks

- Rate = 1/ *period*
- Need to check
  - Processing time for process must be < process deadline
  - Total processing time per cycle must be < period

…Much harder with multiple tasks, periods, and deadlines



8

Pre-emptive Real-Time Schedulers

- Rate Monotonic (RM)
  - Fixed, repeating schedule for worst possible case …*pre-emptive version of last slide*
  - Priority inversely proportional to required work period …*more frequent = higher priority*
  - Assume processes have same amount of work (CPU processing/ burst time) each cycle
- Earliest Deadline First (EDF)
  - Dynamic scheduling scheme
  - Each process has sequence of deadlines …*schedule process with closest deadline*

Recall: we saw EDF with disk scheduling

9

6. File-Systems

Summary/ Recap

10

Overview

- Approaches to storing files
- Common file-systems
  - FAT, traditional Unix, e.g., ext2
- Journaling and Locking
- New approaches

11

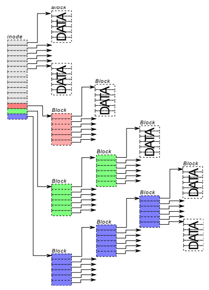
FAT Schemes

- Simple filesystems
- Common on small disks/ consumer products
- Limited capacity
  - Need to scan full sector/ block list to find any point in file
    - List held in File Allocation Table (typically multiple copies for resilience)
  - To be efficient, whole FAT must be in-memory

12

Unix index-node (inode)

- Combined index scheme
  - Random access – can jump to any point in file
  - Index levels/ depth vary with file size
    - Supports very large files
    - Efficient for small files
- Supports sparse files
  - No need to store empty (all zero) blocks
    - Data blocks or index blocks



13

Creating a Sparse File

```c
const char * message = "Hello World\n";

int
main( ) {
        FILE * stream = fopen( "testfile.txt", "w" );

        if( stream == NULL ) { /* Handle error */ }

        fwrite( message, sizeof( char ), strlen( message ), stream );

        fseek( stream, 1024 * 1024 * 1024, SEEK_SET ); // Make hole

        fwrite( message, sizeof( char ), strlen( message ), stream );

        fclose( stream );
}
```

14

Program Output

```
$ gcc hole.c -o hole
$ ./hole
$ ls -l
total 32
-rwxrwxr-x 1 acs acs      17000 Oct 13 21:35 hole
-rw-rw-r-- 1 acs acs        453 Oct 13 21:34 hole.c
-rw-rw-r-- 1 acs acs 1073741836 Oct 13 21:35 testfile.txt
$
```

~ 1GB in size

15

Program Output II

```
$ gcc hole.c -o hole
$ ./hole
$ ls -l
total 32
-rwxrwxr-x 1 acs acs      17000 Oct 13 21:35 hole
-rw-rw-r-- 1 acs acs        453 Oct 13 21:34 hole.c
-rw-rw-r-- 1 acs acs 1073741836 Oct 13 21:35 testfile.txt
$ du -k *
20      hole
4       hole.c
8       testfile.txt
$ od testfile.txt
0000000   H   e   l   l   o       W   o   r   l   d  \n  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
10000000000   H   e   l   l   o       W   o   r   l   d  \n
10000000014
```

~ 1GB in size

Only 8KB on disk (2 x 4K blocks)

Notice the hole

16

Journaling File Systems

- Examples: NTFS, ext3, …
- File-system updates written as *transactions* to journal/ log
  - Transactions periodically flushed to disk
  - Entries only removed from log when confirmed written/ flushed
- Fast as decouples file writes from disk head movement
  - Flush operation can do in-order sector (block) writes
  - Transaction log could be on faster disk/ media
- Resilient as log can be ‘replayed’ in event of system failure
  - Transactions must be ***atomic*** and ***idempotent***

Atomic: All or no subparts of transaction must be completed
Idempotent: Can be repeated any number of times and still give same result

17

Journaling

- (Circular) Transaction log of filesystem updates
  - Can be replayed in event of system failure
  - If committed all updates must done, else none
  - Depending on filesystem, log can be held on same disk or fast SSD

Start (Metadata)	Update	Update	Update	Commit

18

POSIX File Locking: *fcntl( ), lockf( )*

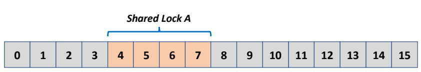
- Often only way to coordinate unrelated applications
- Shared Lock
  - Any number of shared locks possible
  - Cannot include any data byte under an exclusive lock
- Exclusive Lock
  - Fails if any requested byte subject to existing lock
- API allows blocking and non-blocking lock requests
  - Blocking lock request hangs until bytes available (no other locks)

Note: a process can only have one type of lock on a file at any point

Early systems often locked at file descriptor (whole file) level, see (BSD style) flock( ) -- now mainly used for process/ thread synchronisation, but note that threads must explicitly open( ) file to get new descriptor… i.e. not inherit via fork( ), or via dup( ).

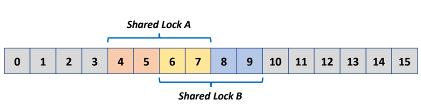
19

POSIX File Locking



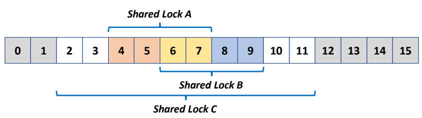
20

POSIX File Locking



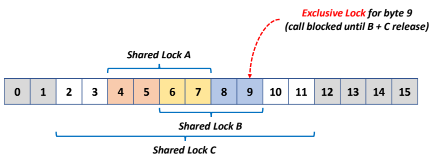
21

POSIX File Locking



22

POSIX File Locking



23

New filesystems: *extents and delayed writes*

- Portions of file stored as contiguous set of disk blocks
  - Makes file reads more efficient
    - Less meta-data (block pointers and indexes, etc.) to process
    - Less fragmentation
    - Less head movement
  - Delay (collate) writes, or applications can use *pre-allocation* with *fallocate( )*
- File information contains list of extents:

Logical Block address (start of extent in file)
Number of contiguous disk blocks in extent
Block address of first disk block in extent

Used in ext4 and similar *Cluster Run* scheme used in NTFS

24

Other Improvements

- Fine grained timestamps
  - Granularity of 1 sec no longer acceptable… use *ns*
- Database technology in filesystems
  - Database style (H or B+ based) indexes for
    - Directories
    - Accessing tree of extents
  - Databases as a filesystem
    - Very powerful
    - Problematic due to semantics of/ expected result common operations
      - No longer simple tree of files and directories expected by users and applications

25

7. Memory Protection

Introduction

26

Memory Protection

- Basic approach
  - Memory Management Unit (MMU)
- Segmentation
  - Variable length scheme
- Paging
  - Fixed length scheme

27