CS2106 Cheatsheet Finals AY2023/2024 Semester 2

Synchronization

Properties of CS

- **Mutual Exclusion** - If a process is within the CS, all other processes are prevented from entering the CS.
- **Progress** - If no process is in the CS, one of the waiting processes should be granted access.
- **Bounded Waiting** - After a process requests to enter the CS¡ there exists an upper-bound of number times other processes can enter the CS before the requesting process.
- **Independence** - The execution of one process in the CS should not affect/block the execution of another process.

TestAndSet Register, MemoryLocation

- Load the current content at MemoryLocation into Register
- Set MemoryLocation to 1
- EnterCS(int *Lock) { while (TestAndSet(Lock) == 1) }
- ExitCS(int *Lock) { *Lock = 0; }

Semaphores

- **Wait(S)**
  If S $\leq$ 0, block the process. Else, decrement S.
- **Signal(S)**
  Increment S. Unblock a sleeping process.
- Imagine list of waiting process.
- $S_{current} = S_{initial}$ − num of processes in CS
- num of processes in CS = $\#Signal(S) - \#Wait(S)$
- General Semaphores can be mimicked using binary Semaphores. S = 0, T = 1.
- generalWait() - if n == 0 then wait(t); wait(s); n–; signal(s)
- generalSignal() - wait(s); n++; signal(s); signal(t)

Producer Consumer

```
while (TRUE) {
        Produce Item;

        wait( notFull );
        wait( mutex );
        buffer[in] = item;
        in = (in+1) % K;
        count++;
        signal( mutex );
        signal( notEmpty );
}
```
Producer

```
while (TRUE) {

        wait( notEmpty );
        wait( mutex );
        item = buffer[out];
        out = (out+1) % K;
        count--;
        signal( mutex );
        signal( notFull );

        Consume Item;
}
```
Consumer

busy waiting, canProd, canCons mutexes.

Writer Reader

```
while (TRUE) {

    wait( roomEmpty );

    Modifies data

    signal( roomEmpty );
}
```
Writer

```
while (TRUE) {

    wait( mutex );
    nReader++;
    if (nReader == 1)
        wait( roomEmpty );
    signal( mutex );

    Reads data

    wait( mutex );
    nReader--;
    if (nReader == 0)
        signal( roomEmpty);
    signal( mutex );
}
```
Reader

Potential starvation for writers.

Tannenbaum's Solution

```
#define N 5
#define LEFT ((i+N-1)% N)
#define RIGHT ((i+1) % N)

#define THINKING 0
#define HUNGRY 1
#define EATING 2

int state[N];
Semaphore mutex = 1;
Semaphore s[N];

void philosopher( int i ){
    while (TRUE){
        Think( );
        takeChpStcks( i );
        Eat( );
        putChpStcks( i );
    }
}
```

```
void takeChpStcks( i )
{
        wait( mutex );
        state[i] = HUNGRY;
        safeToEat( i );
        signal( mutex );
        wait( s[i] );
}
```

```
void safeToEat( i )
{
    if( (state[i] == HUNGRY) &&
        (state[LEFT] != EATING) &&
        (state[RIGHT] != EATING) ) {

        state[ i ] = EATING;
        signal( s[i] );
    }
}
```

```
void putChpStcks( i )
{
        wait( mutex );

        state[i] = THINKING;
        safeToEat( LEFT );
        safeToEat( RIGHT );

        signal( mutex );
}
```

```
void philosopher( int i ){

    while (TRUE){
        Think( );
        wait( seats );
        wait( chpStck[LEFT] );
        wait( chpStck[RIGHT] );
        Eat( );
        signal( chpStck[LEFT] );
        signal( chpStck[RIGHT] )
        signal( seats );
    }
}
```

Memory Management

Continuous Memory Management

Memory Partitioning

- Fixed-size partitioning. Internal Fragmentation. Partition Size need to be large enough to hold largest process.
- Variable-size partitioning. External Fragmentation. Flexible. Hole Merging. Compaction. Additional bookkeeping.

Dynamic Partitioning Algo - First Fit, Best Fit, Worst Fit, Next Fit. Merge when there are adjacent holes .

Buddy System

- Array of linked lists of free blocks of size 2^{index} indicated by start address.
- **Allocate** - Find smallest block that fits. If present, remove from list. If not, find smallest larger free block to split.
- **Free** - Add block to free list. Check buddy. If buddy is free, merge recursively.
- **Find Buddy** - xxx00... and xxx10... are buddies. Same index-1 bits, different $index^{th}$ bit.

Disjoint Memory Management

Paging

- Physical memory divided into fixed-size blocks called frames. (Disjoint)
- Logical memory divided into same-size blocks called pages. (Continuous)
- Page Table: Index = Page Number, Value = Frame Number. Per Process in PCB.
- Physical Address = Frame Number · Page Size + Offset.
- Offset bits = $\log_2$(Page Size). Use remaining bits to find frame from page table.
- Potential Internal Fragmentation at last page.
- 2 Memory Access - Page Table Access, Memory Access.
- TLB - Cache for Page Table. Faster access. — Page# — Frame# —
- Hardware context. TLB Flush on context switch.
- Average Mem Access Time = TLB hit + TLB miss = P(TLB hit) · (TLB access + Mem Access) + P(TLB miss) · (2 * Mem Access).

Segmentation

- Text, Data, Heap Stack. Each segment has a base and limit register.
- Segment table: Index = Segment Number, Value = Base Address, Limit.
- <Segment Number, Offset> → <Base + Offset>. Check if Offset $\leq$ Limit.
- External Fragmentation

Segmentation with Paging

- Each segment has its own page table.
- Segment Table: Index = Segment Number, Value = Page Table Base Address.
- Page Table: Index = Page Number, Value = Frame Number.
- <Segment Number, Page Number + Offset>

Virtual Memory

- Extended Paging Scheme - Memory Resident. Non-Memory Resident.
- Page Fault - Page not in memory. OS loads page from disk.
- Demand Paging - Only load pages when needed.

Page Table Structure

- Page Table takes up memory. Large page table for large address space.
- Number of pages = $\frac{2^{\text{Virtual Address Bits}}}{\text{Page Size}}$
- Size of page table = Number of pages · Page Table Entry Size
- Branch Factor = $\frac{\text{Page Size}}{\text{PTE Size}}$
- Inverted Page Table (Reverse Mapping) - Index = Frame Number, Value = Process ID, Page Number. Auxilary Structure. Linear Scan. Size = Number of Frames · PTE Size.

Multi-Level Paging

- Page Directory pointing to chunk of page tables. 2 Level Paging.
- <Page Directory Index, Page Table Index, Offset>
- Entries per chunk = $\frac{\text{Page Size}}{\text{PTE Size}}$
- 3 Memory Access - Page Directory Access, Page Table Access, Memory Access.
- TLB can be used. But TLB miss is more expensive.

Page Replacement Algorithms

- OPT (Belady's Algorithm) - Replace page that will not be used for longest time. Next use time (max).
- FIFO - Replace page that has been in memory the longest. Initial arrival time (min).
- LRU - Replace page that has not been used for longest time. Last use time (min). Stack. Logical Time Counter
- Clock - Circular list. Hand points to page to be replaced. R++ if hit. If R = 0, replace. Else, set R = 0.

Frame Allocation

- Equal Allocation. Proportional Allocation
- Local Replacement (Local Thrashing). Global Replacement (Cascading thrashing)
- Working Set Model - Δ = Time Interval. WSS_i = Working Set Size. $\tau = \Delta$.

File System

Self-Contained, Persistent, Efficient.

MetaData

- Name. File Extension
- Type. Regular File (ASCII, Binary), Directory, Special File (Device, Pipe, Character/Block oriented)
- Protection. RWX for Owner, Group, Public. *chmod 760 example.txt*
- Structure. Array of Bytes. Fixed Length Records. Variable Length Records. Tree.

Data Access

- Sequential Access. (Contiguous → Linked List → Indexed)
- Random Access.(Contiguous → Indexed → Linked List) — Read(offset), Seek(offset), bytes.
- Direct Access (Fixed Length Records, Random Access on Records)
- Operations: Create, Open, Read, Write, Reposition, Truncate, Close
- open("file.txt", O_RDONLY — O_CREAT, 0644). Returns file descriptor,fd (int).
- read(fd, buffer, size). write(fd, buffer, size). close(fd).

File Information

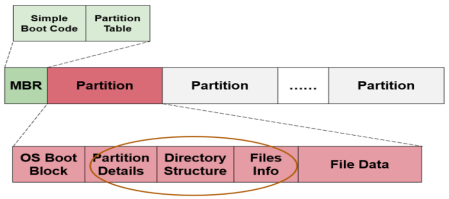
- File Pointer, File Descriptor, Disk Location, Reference Count
- Per-Process Open File Table (In PCB, File Descriptor Table, Points to System-wide Open File Table)
- System-wide Open File Table (Points to Inode Table)
- Inode Table (Points to Data Blocks)
- 0 stdin, 1 stdout, 2 stderr. Redirect pointers with dup2(oldfd, newfd).
- Different Process opening the same file have different fd, different system-wide open file table entry, same inode table entry.
- Parent Child Process. Different fd, same system-wide open file table entry, same inode table entry.

Directory

- Single Level Directory (All files in one directory)
- Tree-Structured Directory (Hierarchical, has subdirectories)
- Directed Acyclic Graph Directory (Multiple parents, Hard Link: Pointers to same inode, hard delete when reference count = 0)
- General Graph Directory (Cycles, Soft Link: Special Link File with pathname to original file, soft delete when file name is deleted)

File System Implementation

Generic Disk Organization

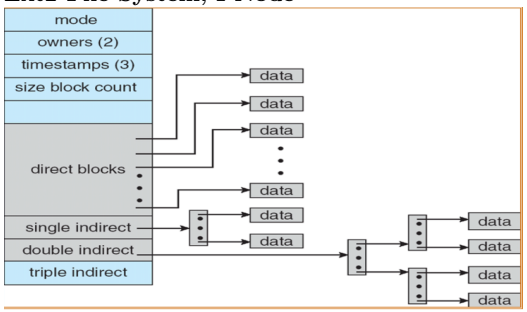


OS Boot Code: Loads OS into RAM. Partition Details: Free Disk Block Management. Directory Structure. Files Info: Metadata. File Data (File Content).

File Data

- Contiguous: (Start address, size). Fast. External Fragmentation. File Size $\leq$ Disk Size.
- Linked List: (Start address, pointerNext, end address). No External Fragmentation. Slow. Pointer overhead.
- Indexed: (Block Location, Index Block Sequence). Index Block: Array of pointers to data blocks. No External Fragmentation. Fast. Index Block[N] Nth Block Address. Extensions: Linked List of index blocks, Multi-level index, Combined Scheme (direct indexing + multi-level indexing)
- File Allocation Table (FAT): Index: Block Number, Value: Next Block Number. -1 = EOF. Can have empty entries. Entries: FREE, Block Num, EOF, BAD.
- FAT16: 2^{16} data blocks in disk. 2^{16} FAT Entries. Each entry is 16 bits.
- FAT is loaded into RAM, Faster Random Access.

Ext2 File System, I-Node



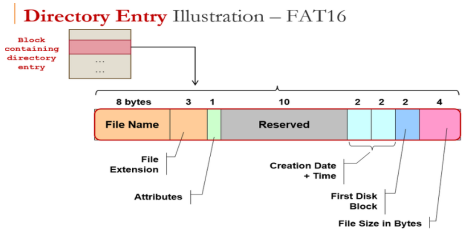
- 12 Direct Blocks. 1 Single Indirect Block. 1 Double Indirect Block. 1 Triple Indirect Block.
- Fast access to small files. Flexibility in handling big files.
- Direct = 12 * Block Size.
- Num of Entries = Block Size in bytes / bytes of block address.
- Single Indirect = Num of Entries * Block Size.
- Double Indirect = Num of Entries^2· Block Size.
- Triple Indirect = Num of Entries^3· Block Size.

Free Space Management

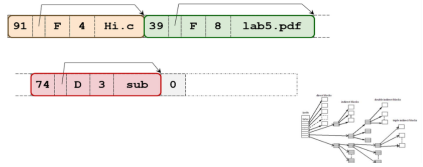
- Partition Details.
- BitMap. 0 = occupied. 1 = free. 1 bit per block. Fast. Space Overhead.
- Linked List. Pointer to next free block.

Directory Structure

- Linear List of Files. (File Name, File Information or pointer to File Information). (File, Start, Length). Linear Search. Cache.
- Hash Table. (Hash Value, File Name, File Information(start, length)). Hash Function. Collisions. Hash Table Limited Size.
- **Directory Entry** Illustration – FAT16



- **Directory Structure – Ext2 FS** (Illustration)



- Directory Permissions. Think of directories as a file with list of files. RWX. ls (R), cd (X), any changes to the "file list" (renaming, add/delete files) (W). Individual file permissions are not affected.

File Operation Walkthrough

Create - Use full pathname to locate parent directory, search for filename. Use free space list to find free disk blocks. Add Entry to parent directory.
Open - Use full pathname to locate file. Add entry to per-process open file table, link pointers to system-wide file table and I-node table. Return file descriptor.

open(fd) and fork() behaviour

File opened before fork(). Child inherits file descriptor. If both parent and child read, both move pointer and result is undeterministic.
File opened after fork(). Parent and child have different file descriptors. Each read their own number of bytes.
File opened before fork(), but buffer has read 50 words before fork(). After fork(), buffer is copied, both parent child read same 50 bytes, afterwards all undeterministic.
File opened before fork(). If read is done using buffer, buffer size number of bytes are not interleaved.