CS2106 Live Class: Synchronization

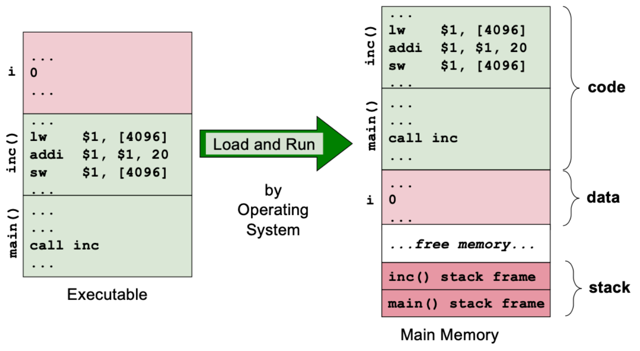
$_{/}|/|U_{C}h@NgRu!$

May 10, 2021

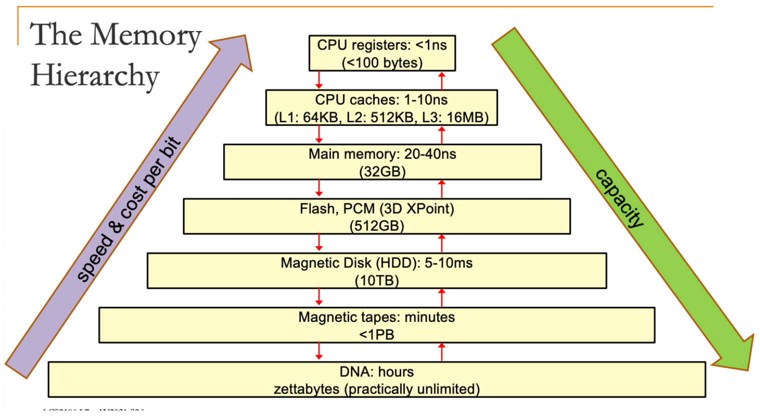
1 Memory hardware

1.1 Physical memory storage

1. Random accesss memory(RAM)
   The minimum addressable address unit is byte
2. can be treated as an array of bytes
3. Each byte has a unique index(the index is the address)
4. Contuguous memory allocation
   (a) fixed size partition
   (b) variable size partition



For process there are 4 regions: text,data,heap and stack. Out of these 4, how text and data are layed out is decided by compiler in executable
For example, it's the compiler that decides the 4096 store the data of i in the above image


The Memory Hierarchy
CPU registers: <1ns (<100 bytes)
CPU caches: 1-10ns (L1: 64KB, L2: 512KB, L3: 16MB)
Main memory: 20-40ns (32GB)
Flash, PCM (3D XPoint) (512GB)
Magnetic Disk (HDD): 5-10ms (10TB)
Magnetic tapes: minutes <1PB
DNA: hours zettabytes (practically unlimited)
speed & cost per bit
capacity

In most case, the OS has no access to the cache(it's hardware's business); The OS is just in charge of which process has the right to accesss the memory and which memory the process has right to access

There are two types of data in a process:

1. **Transient data:** valid only for a limited duration, e.g. function parameter, loval variable
2. **Persistent data:**valid for duration of program unless explicitly removed, e.g. global variable, constant variable, dynamically allocated memory

IMPORTANT: both transient data and persistent data can grow/shrink during execution

2 Memory abstraction

2.1 Reasons of memory abstraction

1. hardware memory access may be conflicted between different processs(what if two process both wanna declare a variable named i)
2. abstraction of memory benefits the protection of mamory space(One process write on the data of the other process(What if the process is OS or some other very important process))
3. more portable(hide details of hardware)

2.1.1 pro of without memory abstraction

Faster, efficient

One simple abstraction is to use **base and registers**

2.2 Address relocation

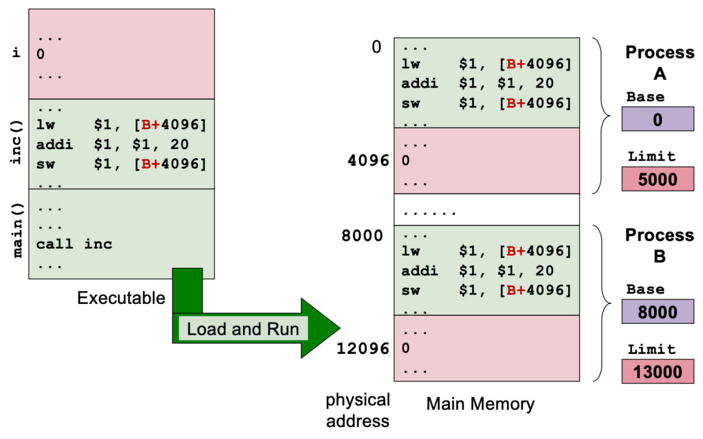
Recalculate the memory references when the process is loaded into memory:
e.g. Add an offset of 8000 to all memory refernces in Process B
The problem:

1. Slow loading time
2. Not easy to distinguish memory reference from normal integer constant

3 base and limit registers

Use a special register as base of all references, which is known as base register. During compilation time, all memory references are compiled as offset from the base register
At the loading time, the base address is initialized as the starting address of the process memory space.
Use another special register to indicate the range of the memory space of the current process, which is known as **limit register**. All memory access is checked against the limit to protect integrity
Disadvantage of base and register: Slow loading time: each time accessing memory, need to do the below calculation

1. Actual= Base + Adr(offset) to get actual address
2. Check Actual < Limit for validity



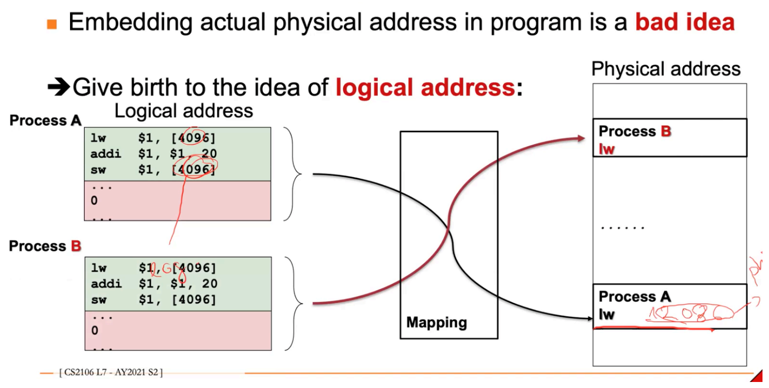
```
...
lw    $1, [B+4096]
addi  $1, $1, 20
sw    $1, [B+4096]
...
```

```
...
...
call inc
...
```

The idea evolves to be the idea of **logical address**, which has a bijection to physical address. Building on this abstraction, each process will have a self-contained, independent logical memory space (The definition of self contained refers to something or someone that is complete on its own and that doesn't need anything else)

4 Logical Address

Embedding actual physical address in program is a bad idea



- Embedding actual physical address in program is a **bad idea**

➔Give birth to the idea of **logical address**:

Process A

```
lw    $1, [4096]
addi  $1, $1, 20
sw    $1, [4096]
...
0
...
```

Process B

```
lw    $1, [4096]
addi  $1, $1, 20
sw    $1, [4096]
...
0
...
```

The synchronization and conflict between process may affect

5 Continuous memory management

Purpose: Multiple processes are allowed to have physical memory at the same time to support context switch. In this section, it is assumed that

1. Each process occupies a **continuous memory regions**
2. The hardware memory space is large enough to support one or more process to have their complete continuous memory region

5.1 Actions when the memory is full

1. removing terminated process
2. swapping blocked process to secondary storage

6 Memory partition

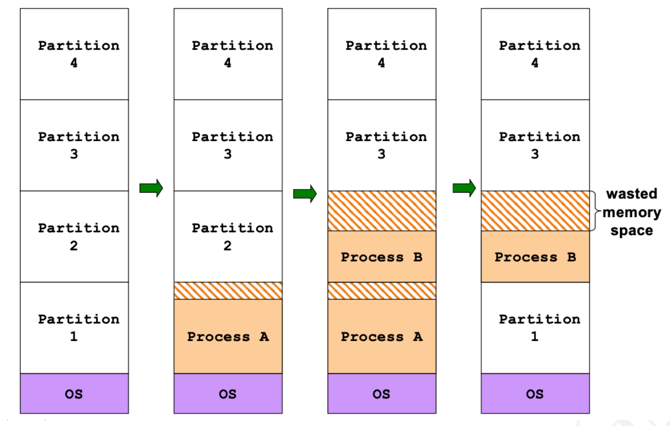
Memory partition is the contiguous memory region allocated to a single process
We can make these partitions in two ways

1. fixed-size partition

2. variable-size partition

6.1 Fixed sized partition

For fixed sized partition, physical memory is split into **fixed number** of **qually sized** partition and a process will occupy one of them
It has the advantage of **easy management** and **fast allocation**. However, partition size need to be large enough to contain the largest of the process
The smaller process will waste space, which is known as internal fragmentation



6.2 Dynamic partition

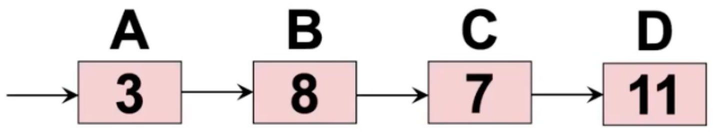
For variable-sized partition, partition is created based on actual size of process. OS **keep track** of the state of occupied and free memory regions and perform **splitting and merging** when necessary
It is **flexible** and **avoid internal fragmentation**. But OS needs to maintain more information and it takes more time to locate appropriate region. Also, external fragmentation occurs, which creates a large number of holes(the small memory can be merged in some ways)
If the OS maintain a list of partitions and holes as below, there is one process need 6, which partition should be selected

6.2.1 The dynamic partitioning types

1. First fit: take the first large-enough hole
2. Best fit: find the samllest hole that is large enough
3. Worst fit: find the largest hole

After finding the target hole with N(N>M, where M is what needed for the process), split. into

M and N-M



1. A: cannot, not large enough
2. B: ralative tight,(first fit)
3. C: the most tight, though the external segmentation is the smallest, the left 1 seg likely to be useless(best fit)
4. D: the external is the largest, but the remains 5 likely to be used later(worst fit)

6.2.2 Free of partition

When occupied partition is freed, algorithm will merge it with adjacent hole if available. Suppose the holes are fragmented, **compaction** can be used to move the occupied partition around to create consolidate holes(Cannot be invoked too frequently coz it;s very time consuming)
The OS usually stores the memory information into a linked list of 3-tuple: (Status, Start Address, Length)

6.2.3 Time complexity

	First fit	Best fit	Worst fit
Allocation	O(1) ∼O(N)	O(N)	O(N)
Deallocation	O(N)	O(N)	O(N)
Merging	O(1)	O(1)	O(1)

6.2.4 Memory deallocation

Deallocation of Memory. Deallocation of memory by the Operating System (OS) is a way to free the Random Access Memory (RAM) of finished processes and allocate new ones. We all know that the computer memory comes with a specific size.

7 Theorem: Buddy System

Buddy memory allocation provides efficient

1. Partition Splitting
2. Locating good match of free partition
3. Partition de-allocation and coalescing

Suppose memory is of 2^K bytes, we keep an array of A[0..K]. Each array element A[j] is a linked list which keep track of free blocks of size 2^j. Each free block is indicated by stating address

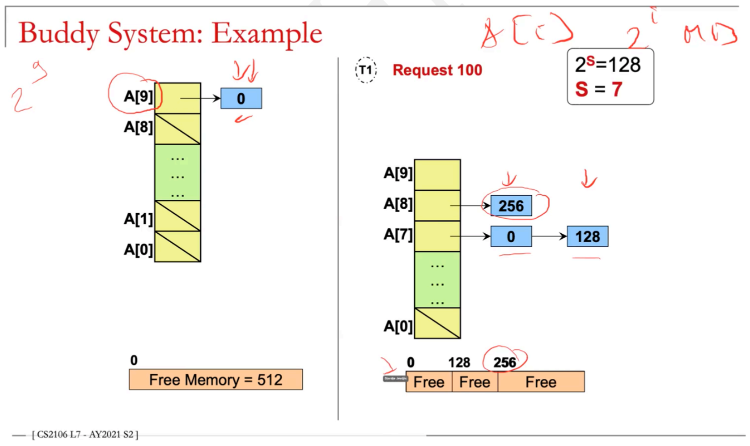
7.1 To allocate a block of size N

1. Find smallest S such that $2^S \geq N$
2. Access A[S] for a free block
   (a) If free block exists, remove block from A[S] and allocate
   (b) Else, find the smallest R from S+1 to K, such that A[R] has a free block B. Reaptedly split B until there are free blocks in A[S] and allocate

7.2 To deallocate a block of size N

1. Check in A[S], for the buddy of B, say C. If C exists, remove B and C, and merge them to B', and try deallocate B'
2. Else, intert B in A[S]

7.3 Example



1. There is a free memory=512;

2. when a new process needing 100 memory comes
3. the A[0]=512 is too big, so break it into two part of parition with 256, one starting at 0 the other start at 256
4. because 256 is still too big, break the first partition at A[8] (the one start at 0)into two part with 128(i.e. A[7]), one start at 0 and the other start at 128
5. the first partition at A[7] is allocated to the process, note here are 28 internal segmentation

7.4 Buddy block

B and C are buddy of size 2S, if

1. The S^{th}Bit of B and C is a complement
2. The laeding bits up to S^{th} bit of B and C are the same

Given block address A is xxxx00..00_2
Get 2 blocks of half the size after splitting:

1. B= xxxxx0..00_2,(The starting position)
2. C= xxxxx1..00_2,(The starting position)

E.g
A= 0 (000000_2),size = 32
After splitting:

1. B=0 (**00**0000_2), size=16
2. B=16 (**01**0000_2), size=16

7.5 Time complexity

	First fit	Best fit	Worst fit	Buddy system
Allocation	O(1) ∼O(N)	O(N)	O(N)	O(logN)
Deallocation	O(N)	O(N)	O(N)	O(N)
Merging	O(1)	O(1)	O(1)	O(logN)