COP290 CLASS3

Tuesday, 14 February 2023 5:45 PM

Call by value / Call be reference

↳ &(variable name)
reference of location

↳ modifications are not available / shown outside while in reference they are still visible outside

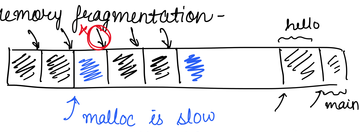
2. Why reference is faster ?
Instead of devoting large memory for saving data again we only save 8 bytes corresponding to reference

4096 pointers → linked lists
↳ basic mapping so collisions are sorted, dont use
↳ check whether collisions are taken or care of i.e. get 2 entries on linked list.
↳ Hash functn aren't being judged

2. Why heaps are worse than stacks ?

↳ memory leaks, stacks do automatic memory management, local variables are reclaimed automatically, For heaps you'll have to allocate free to memory by yourself so you might run out of memory

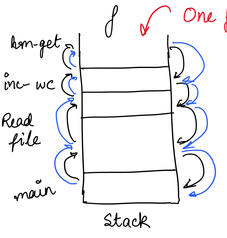
↳ Memory fragmentation -

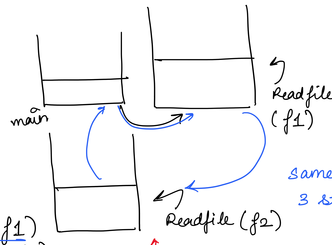


malloc is slow and finds this space where this empty space is found and filled.

→ Heaps on the other hand just leave the space free

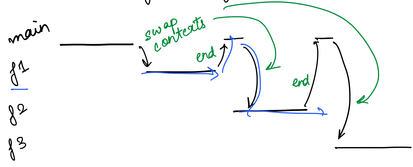
Used or large memory allocation





mythread-create (Readfile f1)
mythread-create (Read file f2)
mythread-join ( )
Many files
gives control to main context

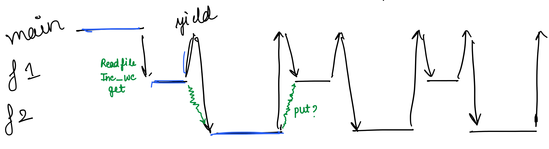
Each stack for new file



Basically complete whole file reading part then next file

funct^n call ~ swap contexts
funct^n return ~ end context

Here contexts are not run to completion

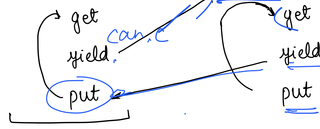


Keep in linked list (f1 f2 f3 f4 ..... → This runs word by word on files / switches files

pointer maintain about where we are
yield call ~ swap context (basically walking around list)

mythreadyield
aquire bucket
Release bucket

→ Hashmap maintains count



- Incorrect without locks
- Concurrently both contexts try to run the read file

(Race condition)