Why NachOS?

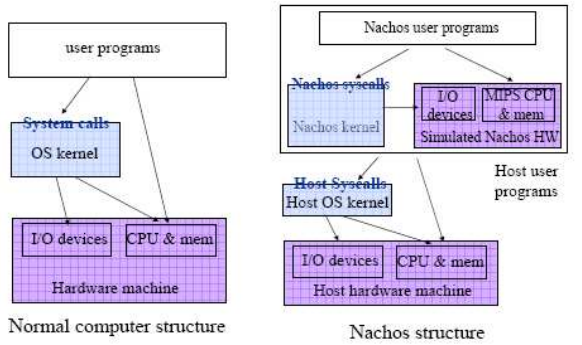
-develop from scratch on real hardware

too complex,

hard enough just to start the machine

-adjust existing OS

lacking a brief view of whole operation system



Main drawback of NachOS

Kernel run on real machine while user program run on simulated machine

So not using the same CPU and same memory

Why not changing it?

Make projects too complex

Hard to debug kernel part

Basic NachOS toolkit

Simulated hardware(MIPS, memory, I/O)

Cross compiler complie C user program to MIPS executable

Limited synchronization primitives (only semaphore)

Only Halt system call

Single user program

Job is to augment it

Do not modify the simulated hardware and compiler

Strengthen the skeleton step by step

-Synchronization
-Multiple user program
-Virtual memory
-File system and disk scheduling

Work on NachOS

Code in C++

Run an executable:
./nachos –x [executable]

Debug
./nachos –d [debug flag]
DEBUG function – very useful

Nachos files

Threads/	thread.cc	thread data structure
	scheduler.cc	manage threads run
	synch.cc	synchronization routine
	boundedbuffer.cc	for bounded buffer
	list.cc	list manage
	synchlist.cc	list with lock
	system.cc	kernel data structure
		startup/shutdown
	utility.cc	

Nachos files

Machine/	disk.cc	
	interrupt.cc	
	machine.cc	
	timer.cc	
	stat.cc	
		should not change here
		keep reference files here

Nachos files

Userprog/	addrspace.cc	addresssspace management
	bitmap.cc	bit map class useful for page management
	exception.cc	handle exec, exit, and fork
Test/	easy C test program, using cross compiler	

Nachos files

filesys/	directory.cc	
	filehdr.cc	file header operation
	filesys.cc	manage overall file system
	openfile.cc	open, read, close file
	synchdisk.cc	synchronized read/write sector
Disk file:	Nachos.xml	
	Seagate.xml	
	IBM36G.xml	

Project 1 threads and synchronization

Add mutex lock and condition variable

Implement a thread-safe bounded buffer class

Implement an alarm clock

Implement an elevator (useful for disk scheduling)

Project 1 a place to get start

Synch.cc

```
void
Semaphore::P()
{
    IntStatus oldLevel = interrupt->SetLevel(IntOff);          // disable
interrupts

    while (value == 0) {                        // semaphore not available
        queue->Append((void *)currentThread);  // so go to sleep
        currentThread->Sleep();
    }
    value--;                                                  // semaphore available,
                                                              // consume its value

    (void) interrupt->SetLevel(oldLevel);    // re-enable interrupts
}
```

Project 1 a place to get start

Synch.cc

```
void
Semaphore::V()
{
    Thread *thread;
    IntStatus oldLevel = interrupt->SetLevel(IntOff);

    thread = (Thread *)queue->Remove();
    if (thread != NULL)        // make thread ready, consuming the V
immediately
            scheduler->ReadyToRun(thread);
    value++;
    (void) interrupt->SetLevel(oldLevel);
}
```

Project 2 multiple user program

-Address space management

A memory manager to allow kernel to allocate frames for processes, keep track of which frame is in use

Set up process page table correctly

-Process management

Handle Exec, Exit and Join system call

Exec- fork and exec

Exit and Join

Project 3 virtual memory

-Page faults

Fault exception happen once accessing invalid Page Table Entry

-Page Replacement

Evict victim page

Save dirty page in backstore

FIFO/LRU

Project 4 file system and disk I/O

-Basic file system

Synchronization between threads on file operation

Caching

File size: multilevel indexed file allocation

-Prefetching

-Disk scheduling