David DeFlyer
17651118
Class notes – CS162
January 26th, 2009

1. Class opening:
    1. Handed out ACM membership information
2. Review of last lecture:
    1. operating systems were something of an ad hoc component
    2. in the 1960s IBM tried to produce a OS for all customers
        1. was an overreach
        2. called OS360
        3. realized that there was a problem when all the manuals wouldn't fit on one desk
    3. all the operating systems of these times were batch system, no time sharing
    4. at the time, this was cost effective because of the cost of the computers compared to programmers
        1. a 2 million dollar machine for four years was much more expensive than a programmer making $7 an hour
    5. in the early 70s, they started to make time sharing systems where multiple people could sit at a terminal and use the same computer
    6. CTS became one of the big ones
    7. IBM tried again to make the best, tried to make TSS, time pressure made them putout TSO while still working on TSS, this cause IBM a lot of trouble because lots of businesses used TSO and didn't want to give it up when TSS came out. TSS was cheaper for the business to use (used less CPU time)
3. Time sharing
    1. made people more productive
    2. helped a lot by the better filling systems instead of decks of punch cards
4. minicomputers:
    1. much cheaper than the bigger computers, only $8000 for a PDP-8
    2. made it a lot easier for small businesses and universities to do computing
    3. had similar computing power as a mainframe
    4. at Berkeley in this time, they were sharing one system that had a load average of 40!
5. UNIX
    1. tried to make a simpler version of the mainframe
    2. started as about 10,000 lines of code
    3. other UNIX systems evolved from this
    4. gave up many of features of mainframes for simplicity
    5. much smaller footprint because of this
    6. UNIX was developed more as a software development platform than as a data processor or storage system
6. PCs
    1. started in the mid 70s with the 8 bit processor, pretty limited
    2. then in 83 they came out with the 16 bit processor with changed a lot of things, made PCs much more useful
    3. much cheaper than the previous mainframes
    4. the evolution of the chips made some things really weird, modern x86 chips can run 8 bit software

David DeFlyer
17651118
Class notes – CS162
January 26th, 2009

5. cisc vs risc
    1. in the 70s they thought that a complex set of instructions would simplify the software, this was not so true
    2. these chips were very, very complex, lots of weird instructions including operating systems and string instructions
    3. a lot of these can't really be removed because of legacy issues
6. IBM also wanted to have multiple sources of compatible chips, so that helped with the x86
7. they have the advantage of being yours and are on your desk
8. they have a lot of disadvantages, they were 10-15 years behind mainframes
    1. not really true now

7. modern operating systems
    1. original UNIX was 10,000 lines
        1. Windows NT was 20 Million
        2. Windows 2000 and XP are 40M
        3. 1000-10000 man years to reproduce
    2. they are very complex
        1. abstract the hardware
        2. conflicting needs
        3. needs to be high performance
    3. they are poorly understood
        1. no one can understand 40 million lines of code
        2. the original designers got promoted and retired
        3. never fully debugged
        4. hard to know exactly how the system is going to react
8. compilers
    1. they were pretty good even in the 60s and 70s

History of Operating Systems ending, starting technical stuff

1. The operating system does many things
    1. one of the big things is to allocate resources
    2. schedule tasks, IO, etc
    3. sharing and protection of resources also
    4. It also can be seen as an extended machine
        1. so you don't have to interact with the PC by flipping switches and looking at flashing lights
        2. you get a nice library, POO (principles of operation), file system, etc
    5. the next month will be about processors, scheduling, coordination, etc.
2. concurrency
    1. the OS makes the programs see things like they are the thing on the machine
3. I/O devices
    1. a long time ago, they CPU would issue I/O commands and would wait till it returned
    2. now interrupts allow the OS to go and do other things and not have to keep polling for the results of the I/O
    3. this adds a lot of complexity, can be very hard to get right

David DeFlyer
17651118
Class notes – CS162
January 26th, 2009

4. memory
    1. need to manage the allocation and use of it
    2. protects other programs
    3. swaps things in and out to make it look like you have more memory
5. files
    1. OS manages the files, helps access
6. Networks
    1. more I/O

4. Coordination
    1. some things need lots of resources, so it can be hard to keep happy
        1. the OS needs to make it look like each application is working alone so that people can write programs more simply, don't have to share resources (from the programmer's POV)
    2. makes a single memory look like a bunch of smaller independent memories
    3. this is all a for of decomposition, chop up a set of hard problems into smaller more manageable tasks
5. processes
    1. is an execution stream in the context of a particular process state
        1. essentially is a running piece of code with its state
    2. process state is everything that can affect of be affected by a process
        1. memory is part of a process's state
    3. execution stream is is a sequence of instructions
    4. not the same thing as a program
        1. a program is the code that is written
        2. a process is the execution stream
        3. compare a script and a play, script is a program, play is a process
    5. uniprogramming systems
        1. really simple systems only allowed one process to be active at once
        2. not to be confused with a uniprocessor system where there is only one processor
    6. multiprogramming systems
        1. is the converse of uniprogramming systems
        2. you can have multiple processes running at once
        3. the processor is able to switch between the different streams of execution
6. switching state
    1. you need to save the state
        1. done in the process control block
        2. registers
            1. general purpose registers
            2. you have system registers also that you don't really see
            3. floating point registers also
        3. not all of memory, though you would store a pointer to a description of the memory state
        4. accounting information such as process time, etc
        5. pointers to open files, file descriptors, etc

David DeFlyer
17651118
Class notes – CS162
January 26th, 2009

        6. processor status
            1. condition codes, system status, etc
    2. you don't want to make the process control block too huge or bulky, so you use a lot of pointers to other structures
    3. organization of control blocks
        1. in a process table
        2. might just be a linked list
    4. all this need to give the view of one CPU per system
        1. this is achieved by switching very quickly between processes
        2. remember the speed that machines execute at
        3. .5 seconds doesn't seem like much to a human, humans have a hard time seeing anything better than .2
7. Scheduler/dispatcher
    1. does:
        1. need to make sure that all processes get to run
        2. need to make sure that processes don't interfere with others
        3. need to make sure that synchronization occurs where needed
    2. decides what to run when and for how long
    3. scheduler and dispatcher are mostly synonyms
        1. IBM has a strange vernacular
            1. the jargon can be very confusing
        2. lots of other companies have their own terms
            1. creates something of a subculture
    4. it runs by
        1. run process
        2. save that state after some time
        3. loads another state
        4. runs that state
        5. repeat
8. how does a processor/dispatcher run?
    1. The process must be ready to run
        1. not blocking on user input or other I/O
    2. need to run the best process if possible
    3. need to chose quickly
9. how do you get control back from the application?
    1. The application gives it back
        1. macs did this at first
        2. requires code to be well behaved
    2. interrupts
        1. forces control back
        2. done by time
10. interrupts
    1. are a type of EIT
        1. Exceptions, Interrupts and Traps

David DeFlyer
17651118
Class notes – CS162
January 26th, 2009

    2. traps are synchronous events where execution can't continue
        1. things like memory error, divide by zero, etc
    2. put you back into a supervisor state
    3. interrupts are asynchronous events on the machine
    4. execution can continue, but something needs attention first
        1. interrupts can be ignored by software if needed
    5. usually, EITs need to go back to the OS because a normal program doesn't have access to the resources needed to deal with these things
11. timers
    1. periodic, ie 60 times/second
        1. every so often it generates an interrupt
        2. not so great because it can produce a lot of overhead
        3. might not be precise enough
    2. time of day
        1. like above, but done by clock time
        2. might be more precise if the machine allows it
    3. elapsed time
        1. interrupt when the timer reaches 0
    4. they are usually only set able in the supervisor mode
        1. a process can call the OS to ask for a timer, the OS then decides whether to allow it
12. context switches
    1. need to be careful not to mess up state
        1. this is because other thread have been working
        2. the OS also has to do some work, so it needs to have some registers in use
    2. this means that the CPU has to be able to support the context switch
        1. some registers have to be both loaded at the same time (PSW)
        2.