Pradip Dhungana
201751
BESE(Day) 5th

AOS (assignment-2)

Q1) List five services provided by an operating system that are designed to make it more convenient for users to use the computer system. In what cases it would be impossible for user-level programs to provide these services? Explain.

⇒ Services:

(I) Program Execution:-
System capability to load a program into memory & to run it.

(II) I/O operation:-
Since, user programs cannot execute I/O operations directly, the operating system must provide some means to perfor I/O.

(III) File-System Manipulation:-
Program capability to read, write, create & delete files

(IV) Communications:-
Exchange of information between processes executing either on the same computer or on different systems t together by a network. Implemented via shared memor or message passing.

(V) Error-detection:-
Ensure correct computing by detecting errors in the CPU & memory

→ Impossible cases for user-level Programs:-

P.T.O
→

(i) User Interface Management:-
→ User-level programs may not have the necessary permissions or system-level control to manage & manipulate the user interface at a fundamental level.
→ Access to hardware resources & low-level system components is typically restricted to the operating system.

(ii) File System Management:-
→ Direct control over file system at a low level may compromise data integrity & security.
→ Operating system enforce permissions & manage file access to prevent unauthorized interface.

(iii) Memory Management:- (Process & Task Management)
→ User-level programs lack the necessary privileges to control the execution of processes & manage the CPU scheduler.
→ Direct manipulation ~~to~~ could lead to resource contention & system instability.

(iv) Security & Access control:-
→ User-level programs cannot enforce access control policies of system level-operating system with their kernel-level control, are essential for implementing & enforcing security measur

(Q2) what are the advantages & disadvantages of using the same system call interface for manipulating both files & devices?

⇒ Advantages:

~~Each device~~ (i) Unified Interface:
Provides a unified & consistent interface for accessing both devices & files. Treating devices as files simplifies user program code, making it easier to interact with both devices & files using the same interface.

(ii) Ease of Development:
Adding new device drivers is relatively easy, requiring implementation of hardware-specific code to support the abstract file interfaces.

(iii) Consistent API:
User program code & device driver code can be written to support a well-defined API, promoting consistency & ease of development.

⇒ Disadvantages:

(i) limited functionality:
It may be challenging to capture the full functionality of certain devices within the context of the file access API, potentially resulting in a loss of functionality.

(ii) Performance Impact:
The use of a common interface might lead to a loss of performance for certain devices, as the file access API

not be optimized for specific device operations.

(iii) Difficulty in Representation:
Representing certain device functionalities within the file system context may be difficult, potentially limiting the expressiveness of device access.

Q.3) Describe the actions taken by a kernel to context-switch between processes.

⇒ The OS must save the PC & user stack pointer of the currently executing process in response to a ~~clock~~ clock interrupt & transfers control to the kernel ~~to~~ clock interrupt handler.

Saving the rest of the registers, as well as other machine state, such as the state of the floating point registers, in the process PCB is done by the clock interrupt handler.

The scheduler to determine the next process to execute is invoked the OS.

Then, the state of the next process from its PCB is retrieved by as & restores the registers. The restore operations take the processor back to the state in which the previous process was previously interrupted, executing in user code with user-mode

privilages.

Many architecture-specific operations, including flushing data & instruction caches also must be performed by context switches.

→ Actions:

(i) Save Current Process Context:
→ Save CPU registers & other relavent state Information.
→ Update the process control Block (PCB) to the current process.

(ii) Update PCB
→ Reflect changes made during the current process's execution.

(iii) Select next Process:
→ Determine the next process to run based on the scheduling algorithm.

(iv) Restore Next Process Control:
→ load the state of the selected process from its PCB

(v) Update memory management.
→ Adjust Memory Mapping if necessary.

(vi) Resume execution:
→ Set up the CPU to execute the selected process.

Q)4) Provide two programming examples in which multithreading doesnot provide better performance than a single-threaded solution.

=> (1) CPU-bounded tasks with (GIL)

Global Interpreter Lock (GIL) in python:

Due to GIL, multithreading may not improved performance significantly for CPU-bound tasks that involve heavy python code execution as only one thread can execute python bytecode at a time.

```
import threading

def cpu_bound_task():
    # Some cpu-bound task
    result = 0
    for _ in range(10**7):
        result += 1

threads = [threading.Thread(target=cpu_bound_task)
for _ in range(4)]

for thread in threads:
    thread.start()

for thread in threads:
    thread.join()
```

(11) limited parallelism in I/O bound tasks:-
for programs primarily I/O bound, where threads spend significantly time waiting for external resources multithreading may not provide substantial performance gains if the I/O operations cannot be effectively parallelized.

```
import threading

def io_bound_task():
    # Some I/O-bound task

with open("example-file.txt", "r") as file:
    context = file.read()

threads = [threading.Thread(target=io_bound_task)
    for _ in range(4)]

for thread in threads:
    thread.start()

for thread in threads:
    thread.join()
```

Q15) Describe the actions taken by a thread library to conte
switch between user-level threads.

=) Actions:-

(1) Save Thread context:
→ The thread library saves the current state of running thread. This includes the value of CPU registers, program counter & any other thread specific info.
→ Thread-specific data structures, such as thread control blocks are updated to reflect the current state of the thread.

(2) Update thread Information:
→ The thread library updates its internal data structures to reflect changes made during the execution of the current thread.

(3) Select the Next thread to Run:
→ The thread library determines the next thread to run based on its scheduling algorithm.
→ This decision is made within the user space, without involving the kernel.

(4) Restore Next Thread context:
→ The thread library restores the state of the selected thread from its saved context.
→ This involves loading the saved register values program counter & other relevant info back into the cpu.

(5) Resume Execution of Next Thread:

→ The CPU is now setup to execute the selected thread. The program counter is loaded with the appropriate address & the thread continues its execution from where it was last interrupted.

User-level thread context switching is handled entirely within the user space without involving the kernel.

Q)6) Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single threaded solution on a single processor system?

⇒ When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaved time in a useful manner.
A single-threaded process, on the other hand, will not be capable of performing useful work when a page fault takes place.
Therefore, in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multithreaded soln would perform better even on single-processor system.

Q)7) Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on single-processor system?

→ A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously.
→ The operating system sees only a single process & will not schedule the diff threads of the process on sperate processes consequently there is no performance benifit associated with executing multiple user-level threads on a multiprocessor system.

Q)8) Discuss how the following pairs of scheduling criteria conflict in certain settings.
a) CPU utilization & response time.
b) Average turnaround time & maximum waiting time
c) I/O device utilization & CPU utilization.

→ a)
→ CPU utilization is increased if the overheads associated with context switching is minimized. Context switching overheads should be lowered by performing context switches infrequently. This could however result in increasing the response time for processes.

→ Average turnaround time & maximum waiting time
Average turnaround time is minized by executing the shortest tasks first such a scheduling po

could however starve long-running tasks & thereby increase their waiting time.

→ I/O device utilization & CPU utilization:
CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they becom ready to run, thereby incurring the overheads of context switches.

Q) 9) What is the meaning of the term busy waiting? Wh Other kinds of waiting are there in an operatin system? Can busy waiting be avoided altogether? Explain ~~your~~ your Answer.

⇒ Busy waiting means that a process is waiting for condition to be satisfied in a tight loop without relinquish the processor. Alternatively, a proces could wait by relinquishing the processor & blo a condition & wait to be awakened at some appr time in the future.

Busy waiting can be avoided by incurs the ove associated with putting a process to sleep & to wake it up when the appropriate program is reached.

(d) 10) Consider the deadlock situation that could occur in the dining-philosophers problem when the philosophers obtain the chopsticks one at a time. Discuss how the four necessary conditions for deadlock indeed hold in this setting.
Discuss how deadlocks could be avoided by eliminating any one of the four conditions.

=> Deadlock is possible because the four necessary conditions hold in the following manner:

1. Mutual Exclusion is required for chopsticks.
2. The philosophers tend to hold onto the chopstick in hand while they wait for other chopstick.
3. There is no preemption of chopsticks in the sense that a chopstick allocated to a philosopher cannot be forcibly taken away.
4. There is a possibility of circular wait.

Deadlocks could be avoided by overcoming the conditions in the following manner:

1) Allow simultaneous sharing of chopsticks.
2) have the philosophers relinquish the first chopstick if they are unable to obtain ~~their~~ other chopstick.
3) Allow for chopsticks to be forcibly taken away if a philosopher has had a chopstick for a long period of time.

4) Enforce a numbering of the chopsticks & always obtain the lower numbered chopstick before obtaining the higher numbered one.

Q) 11) Under what circumstances would a user be better off using a time sharing system rather than a pc or single-user workstation?

=> When there are few other users, the task is large & the hardware is fast, time-sharing makes sense. The full power of the system can be brought to bear on the users problem.
The problem can be solved faster than on a personal computer.
Another case occurs when lots of other users need resources at the same time.
A personal computer is best when the job is small enough to be executed reasonably on it & when performance is sufficient to execute the program to the user's satisfaction.

Q) 12) Ans:
The four necessary condition for a deadlock
(i) Mutual Exclusion
(ii) hold & wait
(iii) no preemption
(iv) circular wait.

The mutual exclusion condition holds as only car can occupy a space in the roadway.

Hold-& wait occurs where a car holds onto their place in the roadway while they wait to advance in the roadway.

A car cannot be removed (i.e. preempted) from its position in the roadway.

Lastly, there is indeed a circular wait as each car is waiting for subsequent car to advance. The circular wait condition is also easily observed from the graphic.

b) A simple rule that would avoid this traffic deadlock is that a car may not advance into an intersection if it is clear they will not be able to immediately clear the intersection.

Q)13) In process creation, what are the possibilities in concerned (1) Parent execution
(2) Address space of the new process (child)

=>-> A process may create several new processes, via a create-process system call, during execution.
-> A parent process creates children processes, which, in turn create other processes, forming a tree of proces

→ Resource sharing, such as CPU time, memory, files, I/O devices...
  ↳ Parent & children share all resources.
  ↳ Children share subset of parent's resources.
  ↳ Parent & child share no resources.

→ When a process creates a new process, two possibilities exist in terms of execution:
  ↳ Parent & children execute concurrently.
  ↳ Parent waits until children terminate.

→ There are also two possibilities in terms of the address space of the new process:
  ↳ child duplicate of parent.
  ↳ child has a program loaded into it.