UPPSALA
UNIVERSITET

# Datorsystem med projekt

Tentaworkshop #2

våren 2020

UPPSALA UNIVERSITET

# 1. Is contiguous memory allocation in general a problem?

...

A, No, there are never issues with contiguous memory allocation for small enough programs.

...

B, Yes, but it is not something to worry about since we have increased speed and efficiency when allocating memory directly as it is which makes it worth the trade off.

...

C, Yes, it will cause fragmentation, meaning that it will not be possible to allocate a program even though we have enough memory.

...

D, Yes, it will cause fragmentation, but this can be solved by chopping the program into small chunks of data. As long as we have enough total empty memory.

UPPSALA UNIVERSITET

# 1. Is contiguous memory allocation in general a problem?

...

A, No, there are never issues with contiguous memory allocation for small enough programs.

...

B, Yes, but it is not something to worry about since we have increased speed and efficiency when allocating memory directly as it is which makes it worth the trade off.

...

C, Yes, it will cause fragmentation, meaning that it will not be possible to allocate a program even though we have enough memory.

...

D, Yes, it will cause fragmentation, but this can be solved by chopping the program into small chunks of data. As long as we have enough total empty memory.

The answer is C. A may be considered correct, but in the long term this will not hold. This is because when removing and allocating new objects we will cause fragmentation at some point. B is poor, vague and not true. D may also be correct, but we cannot directly chop programs into chunks of data. However there are similar techniques for this, such as a page table.

UPPSALA
UNIVERSITET

A process with multiple threads...

(a) ...is created from multiple executables.

(b) ...will terminate even if only one of the threads call exit().

(c) ...is always CPU-bound.

(d) ...shares all of its data across all of its threads.

UPPSALA
UNIVERSITET

A process with multiple threads...

(a) ...is created from multiple executables.

(b) ...will terminate even if only one of the threads call exit().

(c) ...is always CPU-bound.

(d) ...shares all of its data across all of its threads.

For question 2, the correct answer is (b). Despite the presence of multiple threads, the process still counts as a single process, and will therefore be terminated by a single exit() call. The reason that the answer is not (d) is because each thread has a different stack, i.e. stack data is not shared.

UPPSALA UNIVERSITET

John is making a chat application and decides to use sockets for the communication link between the server and the clients. John has heard that TCP is the recommended protocol to use, but is not sure why. **Which of the following statements about TCP is NOT true?**

- TCP makes sure the messages are sent and not lost along the way.
- TCP lowers its sending speed if congestion is detected.
- TCP is generally a lot faster than UDP.
- TCP uses sliding window protocol for reliable delivery.

UPPSALA
UNIVERSITET

John is making a chat application and decides to use sockets for the communication link between the server and the clients. John has heard that TCP is the recommended protocol to use, but is not sure why. **Which of the following statements about TCP is NOT true?**

- TCP makes sure the messages are sent and not lost along the way.
- TCP lowers its sending speed if congestion is detected.
- TCP is generally a lot faster than UDP.
- TCP uses sliding window protocol for reliable delivery.

**Correct Answer:** TCP is generally a lot faster than UDP.

**Motivation:** UDP is generally a lot faster than TCP.

UPPSALA
UNIVERSITET

Which of the following sequence of system calls is the correct way a server using fork operates?
* _use_newSocket_ refers to the resulting child process handling the new connection.

**A.** socket —> bind —> listen —> accept —> close(masterSocket) —> fork —> _use_newSocket_ —> close(newSocket)

**B.** socket —> bind —> accept —> listen —> fork —> close(masterSocket) —> _use_newSocket_ —> close(newSocket)

**C.** socket —> bind —> listen —> accept —> fork —> close(masterSocket) —> _use_newSocket_ —> close(newSocket)

**D.** socket —> bind —> accept —> listen —> fork —> _use_newSocket_ —> close(newSocket)

UPPSALA UNIVERSITET

Which of the following sequence of system calls is the correct way a server using fork operates?
* _use_newSocket_ refers to the resulting child process handling the new connection.

**A.** socket —> bind —> listen —> accept —> close(masterSocket) —> fork —> _use_newSocket_ —> close(newSocket)

**B.** socket —> bind —> accept —> listen —> fork —> close(masterSocket) —> _use_newSocket_ —> close(newSocket)

**C.** socket —> bind —> listen —> accept —> fork —> close(masterSocket) —> _use_newSocket_ —> close(newSocket)

**D.** socket —> bind —> accept —> listen —> fork —> _use_newSocket_ —> close(newSocket)

Correct answer is option C. Debunking the other answers yields; option A closes the master socket prematurely essentially killing the server, option B attempts to accept() before listen() and option D doesn't close the master socket.

UPPSALA UNIVERSITET

I processhantering (Process scheduling) varför vill man ha en medium-term-scheduler?

1. För att man vill kunna hantera vilken av processerna i kön som ska skickas in till CPU:n.
2. För att man vill kunna hantera om en process skall läggas in på hårddisken efter den har skapats.
3. För att man vill kunna hantera om en process skall läggas in i kön efter den har exekverats i processorn eller inte.
4. För att kunna avgöra om processen är klar efter exekvering i CPUn och redo för att termineras.

UPPSALA UNIVERSITET

I processhantering (Process scheduling) varför vill man ha en medium-term-scheduler?

1. För att man vill kunna hantera vilken av processerna i kön som ska skickas in till CPU:n.
2. För att man vill kunna hantera om en process skall läggas in på hårddisken efter den har skapats.
3. För att man vill kunna hantera om en process skall läggas in i kön efter den har exekverats i processorn eller inte.
4. För att kunna avgöra om processen är klar efter exekvering i CPUn och redo för att termineras.

3. MTS används efter en process har exekverats i CPUn och inte termineras, för att avgöra om den ska läggas in på sekundärminnet eller om den ska tillbaka till “ready queuen”, beroende på hur stor andel processer som är CPU-bound gentemot IO-bound.

UPPSALA
UNIVERSITET

A process P1 has two threads T1 and T2. There is a critical section protected by a mutex lock M. First, thread T1 claims the lock M. Thread T2 then uses the `fork()` system call. Which statement is true?

(a) A new process P2 is created, but M is locked forever in P2

(b) A new process P2 is created, containing two threads

(c) A new thread T3 is created, it is able to unlock M

(d) A new thread T3 is created, it cannot unlock M

UPPSALA UNIVERSITET

A process P1 has two threads T1 and T2. There is a critical section protected by a mutex lock M. First, thread T1 claims the lock M. Thread T2 then uses the `fork()` system call. Which statement is true?

(a) A new process P2 is created, but M is locked forever in P2

(b) A new process P2 is created, containing two threads

(c) A new thread T3 is created, it is able to unlock M

(d) A new thread T3 is created, it cannot unlock M

**Correct answer: A.** When `fork()` is called inside a thread, a copy of the process is created, containing one thread, which is a copy of the thread that called `fork()`. This means that P2 contains only one thread, however this thread is unable to unlock M, since only the thread that obtained the lock can unlock it.

Answer B is incorrect since the child process after calling `fork()` only contains the calling thread. (There are alternative versions of `fork()` that copies all threads, however the question specifically states that `fork()` is used.)

Answer C and D are incorrect since `fork()` creates a new process and not a new thread. Answer C is also incorrect since it implies that another thread than the one that has obtained a mutex lock can unlock it.

UPPSALA
UNIVERSITET

When is it desirable to use a multiple process over multiple threads?

a. When a networking program wants to handle several clients
b. Processes have less overhead
c. Context switches are faster when having multiple processes
d. Processes are faster to create and destroy

UPPSALA
UNIVERSITET

When is it desirable to use a multiple process over multiple threads?

a. When a networking program wants to handle several clients
b. Processes have less overhead
c. Context switches are faster when having multiple processes
d. Processes are faster to create and destroy

Multithreading when handling multiple clients is not recommended since threads share data that should only be modified one thread at a time. You have to track multiple threads at a time when debugging which makes it more complicated than when debugging a process. It is hard to code and to interpret results, and is not scalable. Also there is a risk of deadlocks.

The rest of the answers are rather advantages of using threads over processes. Context switching is faster when switching threads compared to process switching because of shared memory. Threads use less resources than processes. Finally, threads are faster to create and destroy.

UPPSALA
UNIVERSITET

Virtual memory gives a program the illusion of a larger memory than it actually has. This is structured in a way that memory is being swapped to and from secondary memory (the disk ex) depending on what addresses programmes want to load. What mechanism takes care of this swapping of data from secondary memory to RAM and vice versa?

1. Dispatcher
2. Mid-term scheduler
3. The page table
4. Short-term scheduler

UPPSALA
UNIVERSITET

Virtual memory gives a program the illusion of a larger memory than it actually has. This is structured in a way that memory is being swapped to and from secondary memory (the disk ex) depending on what addresses programmes want to load. What mechanism takes care of this swapping of data from secondary memory to RAM and vice versa?

1. Dispatcher
2. Mid-term scheduler
3. The page table
4. Short-term scheduler

2. Mid-term scheduler

Motivation: The mid-term scheduler is responsible for swapping out processes from the memory. Swapping is necessary to maintain the process mix. In this case it is usually the oldest piece of data in RAM that gets swapped out to secondary memory (called a page out).

UPPSALA
UNIVERSITET

A unix system uses multilevel queue scheduling, which of the following processes would get the highest priority?

A. A batch process that frequently loads its data from direct blocks
B. A batch process that frequently loads its data from tripple indirect storage
C. An interactive process that frequently loads its data from direct blocks
D. An interactive process that frequently loads its data from tripple indirect storage

UPPSALA
UNIVERSITET

A unix system uses multilevel queue scheduling, which of the following processes would get the highest priority?

A. A batch process that frequently loads its data from direct blocks
B. A batch process that frequently loads its data from tripple indirect storage
C. An interactive process that frequently loads its data from direct blocks
D. An interactive process that frequently loads its data from tripple indirect storage

Motivation: In multilevel queue scheduling processes will gain higher priority the more often they requests IO, thus the interactive processes will gain priority over the batch processes. The IO heavy program that loads data from direct blocks will get its data faster than the one loading data from tripple indirect storage and thus request IO faster, gaining higher priority.

UPPSALA
UNIVERSITET

1. What about processor architecture is **not** true when it comes to threading ?

   (a) Multithreading on a multi-CPU machine increases concurrency.

   (b) A single-threaded process can only run on one CPU, no matter how many are available.

   (c) In a single-processor architecture, more than one threads can be run parallelly.

   (d) Multiple threads may run parallel in multiproccessor architecture

UPPSALA
UNIVERSITET

1. What about processor architecture is **not** true when it comes to threading ?

   (a) Multithreading on a multi-CPU machine increases concurrency.

   (b) A single-threaded process can only run on one CPU, no matter how many are available.

   (c) In a single-processor architecture, more than one threads can be run parallelly.

   (d) Multiple threads may run parallel in multiproccessor architecture

Because single-processor architecture only gives an illusion of running multiple threads when in reality it can only run one thread at one time.

UPPSALA
UNIVERSITET

On a given router ARP entries expire after 60s. Which of the following would be a consequence if the expire time was increased to 2h?

1. The routers' ARP-cache would quickly be maxed out, i.e. it would run out of space.
2. With more entries in the cache the system would become significantly slower.
3. The entries in the cache would have a higher risk of being invalid, increasing the probability of packet loss.
4. All of the above.

UPPSALA UNIVERSITET

On a given router ARP entries expire after 60s. Which of the following would be a consequence if the expire time was increased to 2h?

1. The routers' ARP-cache would quickly be maxed out, i.e. it would run out of space.
2. With more entries in the cache the system would become significantly slower.
3. The entries in the cache would have a higher risk of being invalid, increasing the probability of packet loss.
4. All of the above.

(No motivation given)

UPPSALA
UNIVERSITET

A multithreaded **real-time** online video game has a client program that uses a thread A to update game logic. It also uses a thread B which handles network communication with a server through a socket, relaying any incoming data from the server to thread A.

Thread A also writes to the buffer, which thread B communicates to the server through the socket. Communication between thread A and B is done with the use of two buffers, **buffer_a** and **buffer_b**. A writes to **buffer_a** and reads from **buffer_b**. B writes to **buffer_b** and reads from **buffer_a**. Synchronization is done using a mutex lock for each buffer; only one lock is held at a time by each thread. Neither A nor B lock their respective write buffers if they do not have data to send, but they must lock and read from their respective read buffers to know that they're empty. To avoid indefinite blocking, locks have a timeout period.

Which of the following conclusions can you draw from the information above?

A) UDP is the most preferable when sending any data to the server.

B) Deadlocks may occur.

C) TCP is the most preferable when sending any data to the server.

D) Resource starvation is possible.

UPPSALA UNIVERSITET

Answer: D

Since we're only using mutex locks here, it is possible to end up in a situation where A beats B in acquiring a lock, or vice versa. This could happen multiple times over and over, resulting in starvation in the worst case.

For answer A:

Although online video games are real-time systems (where low network latency is definitely in high demand), it is never stated **what** kind of data is being sent. It could be information of the game state (which *could* benefit from UDP, but it depends on if the information is time critical or not), or it could be transfer of other information, like chat messages, which would benefit more from reliability if users are to communicate properly (TCP or a custom protocol layered over UDP may be beneficial here), or it could be both (in which case both UDP and TCP may both be preferable). There is not enough information to make a sweeping generalization that UDP is the most beneficial in this case. Therefore A is incorrect. The same goes for C.

The Hold and wait condition is not satisfied; it is stated that only one mutex lock can be held at a time. Therefore deadlocks cannot occur. Therefore B is incorrect.

UPPSALA
UNIVERSITET

A process with 32-bit long logical address and 4K byte page size. How many bits are the page offset?

a. 16-bits
b. 8-bits
c. 4-bits
d. 12-bits

UPPSALA
UNIVERSITET

A process with 32-bit long logical address and 4K byte page size. How many bits are the page offset?

a. 16-bits
b. 8-bits
c. 4-bits
d. 12-bits

The correct answer D 12-bits. Page offset is same both in the logical address and physical address. If we have 4K byte page, we need 12 bits to represent the offset.

UPPSALA
UNIVERSITET

Anta att en pipe mellan två processer är implementerad som en bounded buffer, där process A skriver till pipen och process B läser från pipen. Tiden det tar för process B att bearbeta och läsa data från pipen är mycket högre än tiden det tar för A att skriva in data, vilket leder till att buffern fylls väldigt fort.

Anta även att arbetet i process B kan parallelliseras. Skulle det vara möjligt att introducera en process C som är en fork av B för att snabba upp bearbetningen av datan och förhindra att buffern fylls upp?

1. Nej, eftersom deadlock kan uppstå på grund av att implementationen av en bounded buffer endast har två semaforer och ett mutex-lås.

2. Ja, eftersom C öppnar en ny pipe vid en fork som är kopplad till A.

3. Nej, eftersom file-descriptorn för pipen inte kopieras över till C vid en fork och därför kan inte C läsa från pipen.

4. Ja, eftersom implementationen av en bounded buffer med två semaforer och ett mutex-lås tillåter fler än två användare.

UPPSALA UNIVERSITET

Anta att en pipe mellan två processer är implementerad som en bounded buffer, där process A skriver till pipen och process B läser från pipen. Tiden det tar för process B att bearbeta och läsa data från pipen är mycket högre än tiden det tar för A att skriva in data, vilket leder till att buffern fylls väldigt fort.

Anta även att arbetet i process B kan parallelliseras. Skulle det vara möjligt att introducera en process C som är en fork av B för att snabba upp bearbetningen av datan och förhindra att buffern fylls upp?

1. Nej, eftersom deadlock kan uppstå på grund av att implementationen av en bounded buffer endast har två semaforer och ett mutex-lås.

2. Ja, eftersom C öppnar en ny pipe vid en fork som är kopplad till A.

3. Nej, eftersom file-descriptorn för pipen inte kopieras över till C vid en fork och därför kan inte C läsa från pipen.

4. Ja, eftersom implementationen av en bounded buffer med två semaforer och ett mutex-lås tillåter fler än två användare.

4 - Eftersom bounded buffers fungerar med godtyckligt antal användare utan att det uppstår några deadlocks om den implementerats rätt med två semaforer och ett mutex-lås.

UPPSALA
UNIVERSITET

Which of these data structures could be used to help emulate the functionality of a pipe in the kernel?

[ ] Mutex Lock

[ ] File Descriptor

[ ] Semaphore

[ ] Syscall

UPPSALA UNIVERSITET

Which of these data structures could be used to help emulate the functionality of a pipe in the kernel?

[ ] Mutex Lock

[ ] File Descriptor

[ ] Semaphore

[ ] Syscall

En semafor är mycket lämpad för att skapa en bounded buffer, vilket en pipe bygger på för att fungera som den ska. Ett Mutex Lock kanske kan vara relevant, men har inget speciellt med pipes att göra utöver det vanliga: pipes skulle kunna låta bli att använda ett mutex lock utan större svårigheter, men utan en semafor blir det mycket jobbigare. File Descriptors är relaterade till pipes till viss grad, men har inte med pipes funktionalitet i kernelen att göra. Syscall är inte en datastruktur.

UPPSALA
UNIVERSITET

A server programmer is using a multithreading solution to listen for new connections in a server-client-model. Every new connection is assigned to a new thread. Which of the following is **NOT** true?

a. When using threads deadlocks can occur
b. Using threads is not scalable for a large number of clients
c. Threads are easy to code and debug
d. Threads share memory by default

UPPSALA
UNIVERSITET

A server programmer is using a multithreading solution to listen for new connections in a server-client-model. Every new connection is assigned to a new thread. Which of the following is **NOT** true?

a. When using threads deadlocks can occur
b. Using threads is not scalable for a large number of clients
c. Threads are easy to code and debug
d. Threads share memory by default

**Alternative c.** Coding and testing programs on multiple threads is inherently difficult[1].

UPPSALA
UNIVERSITET

What is not true about a pipe?

a) A pipe is a bounded buffer and is implemented solving a typical synchronization problem.

b) Since a pipe is a shared resource by two processes, a write to a pipe must always be an atomic operation.

c) A pipe can only be created using a system call.

d) A pipe is a FIFO buffer that may be used for interprocess communication.

UPPSALA
UNIVERSITET

What is not true about a pipe?

a) A pipe is a bounded buffer and is implemented solving a typical synchronization problem.

b) Since a pipe is a shared resource by two processes, a write to a pipe must always be an atomic operation.

c) A pipe can only be created using a system call.

d) A pipe is a FIFO buffer that may be used for interprocess communication.

b) Since a pipe is a shared resource by two processes, a write to a pipe must always be an atomic operation.

When writing to a pipe with less than PIPE_BUF bytes the write must be atomic since the output data is written to the pipe as a contiguous sequence. But when writing to a pipe with more than PIPE_BUF bytes the operation may be non-atomic. The kernel may interleave the data, on arbitrary boundaries, with data written by other processes. Where PIPE_BUF is defined by each implementation, but the minimum is 512 bytes[1].

UPPSALA
UNIVERSITET

Which of the following is true regarding a scheduler dispatch?

a) A read or write to memory causes a scheduler dispatch.

b) A read or write to non-cached memory causes a scheduler dispatch.

c) A memory read or write only results in a scheduler dispatch if the address translation is not cached (i.e. not cached in the TLB) and the target memory of the read/write is not cached.

d) None of these operations cause a scheduler dispatch.

UPPSALA
UNIVERSITET

Which of the following is true regarding a scheduler dispatch?

a) A read or write to memory causes a scheduler dispatch.

b) A read or write to non-cached memory causes a scheduler dispatch.

c) A memory read or write only results in a scheduler dispatch if the address translation is not cached (i.e. not cached in the TLB) and the target memory of the read/write is not cached.

d) None of these operations cause a scheduler dispatch.

Correct answer: (d)

Motivation: None of the listed alternatives cause a scheduler dispatch since even CPU bound processes access memory frequently, and even the scheduler and dispatcher needs to access memory in order to perform the scheduler dispatch. By the time the dispatch has completed, the initial memory access would be completed.

UPPSALA
UNIVERSITET

**What must happen before the CPU-scheduler choses the next process to be executed?**

a) It most get some form of signal such as an I/O interrupt or clock interrupt.
b) The ready queue has to be full.
c) Only when a process terminates.
d) The process in the cpu must expire its time slice

UPPSALA UNIVERSITET

**What must happen before the CPU-scheduler choses the next process to be executed?**

a) It most get some form of signal such as an I/O interrupt or clock interrupt.
b) The ready queue has to be full.
c) Only when a process terminates.
d) The process in the cpu must expire its time slice

**Solution for the first question is a) this is information obtained from the m3 slides about scheduling which states that the short-term scheduler (CPU scheduler) choses the next process to execute after it has received an I/O interrupt, clock interrupt, an operating system call or another form of signal**

UPPSALA
UNIVERSITET

# INFORMATION OM TENTAMEN

UPPSALA UNIVERSITET

# Information om tentamen

- DEL A
  - 7 stycken sektioner värda 10 poäng vardera
    - Mixed concepts + modul 1 – 6
  - För godkänt på del A och på tentamen krävs:
    - Minst 3 poäng på varje sektion i del A
    - Minst 42 poäng totalt i del A
- DEL B
  - 5 stycken frågor kring ett och samma case, 4p / fråga
  - För betyg 4 krävs minst 10 poäng på del B
  - För betyg 5 krävs minst 15 poäng på del B
  - För betyg 4 eller 5 krävs dessutom godkänt på del A

UPPSALA UNIVERSITET

# Information om tentamen

- E-tentamen
- Inget besök av lärare i skrivsalen
  - Frågor via telefon
  - Korrigeringar via meddelanden
- Del A: Huvudsakligen flervalsfrågor
- Del B: Enbart frågor med fritextsvar
  - Var kortfattad och tydlig

UPPSALA UNIVERSITET

# Om man inte kan/vill skriva tentan

- Omtentamen i Juni samt i Augusti
  - Ej ännu bestämt om det blir e-tentamen eller ej

UPPSALA UNIVERSITET

# Bedömning och betyg

- Flervalsfrågor rättas automagiskt
  - stickprovskontroller
- Fritextfrågor i del B bedöms efter tydlighet och korrekthet
  - okänt vilka möjligheter till återkoppling som finns
- Tentan är endast 1p av kursen
  - Kursbetyget INTE samma sak som tentabetyget
  - Sammanvägd bedömning