Daniel Gooden

COSC240

Assignment 3

Question 1

Total resources:

- Tape Drives: 8
- Modems: 9
- Printers: 8

Available resources:

- Tape Drives: 1
- Modems: 2
- Printers: 1

Need Matrix

Process	Tape Drives	Modems	Printers
P0	0	2	0
P1	0	3	5
P2	8	6	0
P3	2	1	1
P4	1	5	0

To determine a safe sequence we can see which process can currently be finished and allocate resources to it to lead to a safe state.

First we start with process P0. We can allocate the resources required (0, 2, 0) of the available resources. When the process completes we now have the following available resources:

- Tape Drives: 1
- Modems: 5
- Printers: 3

Next we check which process can run with the available resources. P4 is the only available process that can run so we allocate the resources (1, 5, 0) and after the process is finished we now have the following available resources:

- Tape Drives: 2
- Modems: 8
- Printers: 5

Again we check which processes can run. P1 is able to run so we allocate the resources (0, 3, 5) and after the process finishes we now have the following available resources:

- Tape Drives: 3
- Modems: 9
- Printers: 6

We return to P3 and allocate the needed resources (2, 1, 1) and after the process finishes we have the following resources available:

- Tape Drives: 8
- Modems: 9
- Printers: 6

We return to P2 and allocate the needed resources (8, 6, 0) and after the process finishes, we have completed all of the processes required.

The system is in a safe state and the safe sequence is P0 → P4 → P1 → P3 → P2

This ensures that all processes can complete without causing a deadlock.

Question 2

> Consider a process that refers to five pages: A, B, C, D, and E. The pages are referenced in the following order: C, A, C, E, B, D, C, C, B, D, E, A, E, B, D. Assuming memory is initially empty, and that no other processes are running, show the contents of each page frame for each step, along with any other necessary data (i.e. status of the referenced bit for second-chance and clock, process queue for second-chance, and the clock pointer for clock), and determine the number of page transfers during this sequence of references for the following page replacement algorithms and numbers of page frames:
>
> •First-in, first-out:
>
> 1. 3 page frames

Reference	Status	Frame	Queue
C	Page Fault	C, -, -	C

A	Page Fault	C, A, -	C, A
C	In memory	C, A, -	C, A
E	Page Fault	C, A, E	C, A, E
B	Page Fault	B, A, E	A, E, B
D	Page Fault	B, D, E	E, B, D
C	Page Fault	B, D, C	B, D, C
C	In memory	B, D, C	B, D, C
B	In memory	B, D, C	B, D, C
D	In memory	B, D, C	B, D, C
E	Page Fault	E, D, C	D, C, E
A	Page Fault	E, A, C	C, E, A
E	In memory	E, A, C	C, E, A
B	Page Fault	E, A, B	E, A, B
D	Page Fault	D, A, B	A, B, D

10 total page faults

2. 4 page frames

Reference	Status	Frame	Queue
C	Page Fault	C, -, -, -	C
A	Page Fault	C, A, -, -	C, A
C	In Memory	C, A, -, -	C, A
E	Page Fault	C, A, E, -	C, A, E
B	Page Fault	C, A, E, B	C, A, E, B
D	Page Fault	D, A, E, B	A, E, B, D
C	Page Fault	D, C, E, B	E, B, D, C
C	In Memory	D, C, E, B	E, B, D, C
B	In Memory	D, C, E, B	E, B, D, C
D	In Memory	D, C, E, B	E, B, D, C
E	In Memory	D, C, E, B	E, B, D, C
A	Page Fault	D, C, A, B	B, D, C, A
E	Page Fault	D, C, A, E	D, C, A, E
B	Page Fault	B, C, A, E	C, A, E, B

D	Page Fault	B, D, A, E	A, E, B, D

10 total page faults

3. 5 page frames

Reference	Status	Frame	Queue
C	Page Fault	C, -, -, -, -	C
A	Page Fault	C, A, -, -, -	C, A
C	In Memory	C, A, -, -, -	C, A
E	Page Fault	C, A, E, -, -	C, A, E
B	Page Fault	C, A, E, B, -	C, A, E, B
D	Page Fault	C, A, E, B, D	C, A, E, B, D
C	In Memory	C, A, E, B, D	C, A, E, B, D
C	In Memory	C, A, E, B, D	C, A, E, B, D
B	In Memory	C, A, E, B, D	C, A, E, B, D
D	In Memory	C, A, E, B, D	C, A, E, B, D
E	In Memory	C, A, E, B, D	C, A, E, B, D
A	In Memory	C, A, E, B, D	C, A, E, B, D
E	In Memory	C, A, E, B, D	C, A, E, B, D
B	In Memory	C, A, E, B, D	C, A, E, B, D
D	In Memory	C, A, E, B, D	C, A, E, B, D

5 total page faults

•Second-chance (assuming referenced bit is set when a page is first put in memory):

1. 3 page frames

Reference	Status	Frame w/ Ref Bit	Queue
C	Page Fault	C1, -, -	C
A	Page Fault	C1, A1, -	C, A
C	In Memory	C1, A1, -	A, C
E	Page Fault	C1, A1, E1	A, C, E
B	Page Fault	C0, B1, E0	C, E, B

D	Page Fault	D1, B1, E0	E, B, D
C	Page Fault	D1, B1, C1	B, D, C
C	In Memory	D1, B1, C1	B, D, C
B	In Memory	D1, B1, C1	D, C, B
D	In Memory	D1, B1, C1	C, B, D
E	Page Fault	D0, B0, E1	B, D, E
A	Page Fault	D0, A1, E1	D, E, A
E	In Memory	D0, A1, E1	D, A, E
B	Page Fault	B1, A1, E1	A, E, B
D	Page Fault	B0, D1, E0	E, B, D

10 total page faults

2. 4 page frames

Reference	Status	Frame w/ Ref Bit	Queue
C	Page Fault	C1, -, -, -	C
A	Page Fault	C1, A1, -, -	C, A
C	In Memory	C1, A1, -, -	A, C
E	Page Fault	C1, A1, E1, -	A, C, E
B	Page Fault	C1, A1, E1, B1	A, C, E, B
D	Page Fault	C0, D1, E0, B0	C, E, B, D
C	In Memory	C1, D1, E0, B0	E, B, D, C
C	In Memory	C1, D1, E0, B0	E, B, D, C
B	In Memory	C1, D1, E0, B1	E, D, C, B
D	In Memory	C1, D1, E0, B1	E, C, B, D
E	In Memory	C1, D1, E1, B1	C, B, D, E
A	Page Fault	A1, D0, E0, B0	B, D, E, A
E	In Memory	A1, D0, E1, B0	B, D, A, E
B	In Memory	A1, D0, E1, B1	D, A, E, B
D	In Memory	A1, D1, E1, B1	A, E, B, D

6 total page frames

3. 5 page frames

Reference	Status	Frame w/ Ref Bit	Queue
C	Page Fault	C1, -, -, -, -	C
A	Page Fault	C1, A1, -, -, -	C, A
C	In Memory	C1, A1, -, -, -	A, C
E	Page Fault	C1, A1, E1, -, -	A, C, E
B	Page Fault	C1, A1, E1, B1, -	A, C, E, B
D	Page Fault	C1, D1, E1, B1, D1	A, C, E, B, D
C	In Memory	C1, D1, E1, B1, D1	A, E, B, D, C
C	In Memory	C1, D1, E1, B1, D1	A, E, B, D, C
B	In Memory	C1, D1, E1, B1, D1	A, E, D, C, B
D	In Memory	C1, D1, E1, B1, D1	A, E, C, B, D
E	In Memory	C1, D1, E1, B1, D1	A, C, B, D, E
A	In Memory	C1, D1, E1, B1, D1	C, B, D, E, A
E	In Memory	C1, D1, E1, B1, D1	C, B, D, A, E
B	In Memory	C1, D1, E1, B1, D1	C, D, A, E, B
D	In Memory	C1, D1, E1, B1, D1	C, A, E, B, D

5 total page faults

•Clock (assuming referenced bit is set when a page is first put in memory):

1. 3 page frames

Reference	Status	Frame w/ Ref Bit	Queue
C	Page Fault	C1, -, -	C
A	Page Fault	C1, A1, -	C, A
C	In Memory	C1, A1, -	C, A
E	Page Fault	C1, A1, E1	C, A, E
B	Page Fault	B1, A0, E0	A, E, B
D	Page Fault	B1, D1, E0	E, B, D
C	Page Fault	B1, D1, C1	B, D, C
C	In Memory	B1, D1, C1	B, D, C

B	In Memory	B1, D1, C1	B, D, C
D	In Memory	B1, D1, C1	B, D, C
E	Page Fault	E1, D0, C0	D, C, E
A	Page Fault	E1, A1, C0	C, E, A
E	In Memory	E1, A1, C0	C, E, A
B	Page Fault	E1, A1, B1	E, A, B
D	Page Fault	D1, A0, B0	A, B, E

10 total page faults

2. 4 page frames

Reference	Status	Frame w/ Ref Bit	Queue
C	Page Fault	C1, -, -, -	C
A	Page Fault	C1, A1, -, -	C, A
C	In Memory	C1, A1, -, -	C, A
E	Page Fault	C1, A1, E1, -	C, A, E
B	Page Fault	C1, A1, E1, B1	C, A, E, B
D	Page Fault	D1, A0, E0, B0	A, E, B, D
C	Page Fault	D1, C1, E0, B0	E, B, D, C
C	In Memory	D1, C1, E0, B0	E, B, D, C
B	In Memory	D1, C1, E0, B1	E, B, D, C
D	In Memory	D1, C1, E0, B1	E, B, D, C
E	In Memory	D1, C1, E1, B1	E, B, D, C
A	Page Fault	D0, C0, A1, B0	B, D, C, A
E	Page Fault	D0, C0, A1, E1	D, C, A, E
B	Page Fault	B1, C0, A1, E1	C, A, E, B
D	Page Fault	B1, D1, A1, E1	A, E, B, C

10 total page faults

3. 5 page frames

Reference	Status	Frame w/ Ref Bit	Queue
C	Page Fault	C1, -, -, -, -	C

A	Page Fault	C1, A1, -, -, -	C, A
C	In Memory	C1, A1, -, -, -	C, A
E	Page Fault	C1, A1, E1, -, -	C, A, E
B	Page Fault	C1, A1, E1, B1, -	C, A, E, B
D	Page Fault	C1, A1, E1, B1, D1	C, A, E, B, D
C	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D
C	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D
B	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D
D	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D
E	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D
A	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D
E	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D
B	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D
D	In Memory	C1, A1, E1, B1, D1	C, A, E, B, D

5 total page faults

Question 3

Suppose an I-node can hold 11 direct block pointers, 5 indirect block pointers, and 3 double-indirect block pointers. Given a block size of 4KB, and 128-bit block numbers:

•What is the size (in bytes) of the smallest file that requires use of an indirect block pointer?

11 direct block pointers x 4096 bytes = 45056 bytes
The smallest size of file that requires an indirect block pointer is 45056 bytes + 1 byte = **45057 bytes**.

•What is the size (in bytes) of the smallest file that requires use of a double-indirect block pointer?

128 bit/ 8 = 16 bytes
4096 bytes / 16 bytes per pointer = 256 pointers per indirect block
4096 bytes * 256 pointers = 1048576 bytes or 1 Megabyte
So the total size is going to be total amount of bytes in the 11 direct block pointers (determined in part 1) + 5 * 1 MB + 1 byte.
45056 bytes + 5 * 1 MB + 1 byte = **5287937 bytes**

•What is the size (in bytes) of the largest file supported by this system?
Each double indirect block points to 256 indirect blocks, and each of those blocks points to 256 data blocks. So 1 double indirect block can

handle 256 * 1 MB = 256MB
There is 3 double indirect block pointers, the total double indirect box space is: 3 * 256MB = 768MB
The total maximum file size:
45056 bytes + 5MB + 768MB
= 45056 bytes + 5242880 bytes + 805306368 bytes
= **810594304 bytes**

•What is the size (in bytes) of the largest file that would be supported by this system if it had 64-bit block numbers instead of 128-bit block numbers?

The total size of the direct blocks remains unchanged. However the total size of the indirect and double indirect block pointers increases.
The number of pointers per block becomes:
4096 bytes / 8 bytes per pointer = 512 pointers
Each indirect block can handle 512 pointers * 4096 bytes = 2MB
Each double indirect block can handle 512 * 2MB = 1GB

The total maximum file size with 64 bit blocks:
45056 bytes + 5 * 2MB + 3 * 1GB
= 45056 bytes + 10485760 bytes + 3221225472 bytes
= **3231756288 bytes**

Question 4

An application program takes time T to execute on a 10-computer cluster
We can use Amdahl's law to calculate the speed up

Definition [edit]

Amdahl's law can be formulated in the following way:[3]

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

where

- S_{latency} is the theoretical speedup of the execution of the whole task;
- s is the speedup of the part of the task that benefits from improved system resources;
- p is the proportion of execution time that the part benefiting from improved resources originally occupied.

Taken from Wikipedia https://en.wikipedia.org/wiki/Amdahl%27s_law

•What is the effective speedup of running this application on the cluster rather than a single computer if 25% of T is spent running on all 10 computers, and

the remaining time is spent running on a single computer?

$$S = \frac{1}{(1-0.25)+\frac{0.25}{10}} \approx 1.29$$

•What is the effective speedup of running this application on the cluster rather than a single computer if 50% of T is spent running on all 10 computers, and the remaining time is spent running on a single computer?

$$S = \frac{1}{(1-0.5)+\frac{0.5}{10}} \approx 1.82$$

•What is the effective speedup of running this application on the cluster rather than a single computer if 7% of T is spent running on all 10 computers, 34% of T is spent running on 7 computers, and the remaining time is spent running on a single computer?

$$S = \frac{1}{(1-0.07-0.34)+\frac{0.07}{10}+\frac{0.34}{7}} \approx 1.55$$

•What is the effective speedup of running this application on the cluster rather than a single computer if 94% of T is spent running on all 10 computers,4% of T is spent running on 6 computers, and the remaining time is spent running on a single computer?

$$S = \frac{1}{(1-0.94-0.04)+\frac{0.94}{10}+\frac{0.04}{6}} \approx 8.29$$

Question 5

A computer virus is malicious software that replicates itself by modifying other programs to insert its own code. Antivirus software was initially designed to detect and remove computer viruses, but now also provides protection from other potential computer threats. Write a brief (approx. 500 words) history of the development of antivirus software, describing some of the techniques used by viruses that have required new methods of detection for antivirus software.

The origins of antivirus software start in the 1970's when a virus known as the "Creeper Virus". The Creeper virus, was originally created as an experimental self-replicating program that moved from one machine to another, printing the message, "I'm the Creeper, catch me if you can!" Ray Tomlinson then wrote a program that is now referred to the first antivirus called "The Reaper". It's sole purpose was to search for and delete the Creeper virus. This set the stage for the development of software that could detect and remove malicious code from computers.

It wasn't until the 1980's, with the boom in personal computers and rise of widespread viruses, that antivirus software started to take off. The spread of viruses like "Brain" began to get developers to start developing software to deal with the growing threat of viruses. The software of

this era primarily relied on signature-based detection which would compare files on a computer to known virus signatures, looking for the “fingerprint” of a virus. If a match was found, the software would either delete or isolate the file to deal with virus.

As viruses became more complex, antivirus software had to get smarter. By the 1990s, we started seeing polymorphic viruses like "Tequila." These viruses would change their code every time they spread, making it harder for signature-based detection to catch them. To deal with this, developers introduced heuristic-based detection. Instead of just looking for known virus signatures, this method analyzed how programs behaved, flagging anything suspicious. This approach allowed antivirus software to detect new or modified versions of viruses, even if it didn’t recognize the exact code.

The 2000s brought a whole new wave of malware: worms, Trojans, spyware, ransomware, etc. One of the most infamous was the "ILOVEYOU" virus in 2000, which spread through email and infected millions of systems worldwide. As the internet became essential to daily life, cyber threats kept evolving, and antivirus software had to keep up. Real-time behavioral detection became crucial, allowing antivirus programs to monitor what applications were doing and stop threats before they could cause damage. Sandboxing also became popular. This allowed running potentially dangerous files in isolated environments to keep the main system safe.

Today, antivirus software is just one part of a larger security package. These programs now come bundled with firewalls, anti-phishing tools, ransomware protection, and advanced data security features. On top of that, modern antivirus tools use machine learning and artificial intelligence to spot emerging threats faster than ever. By analyzing huge datasets of malware samples, AI can detect patterns that signal malicious behavior, even if the virus is new.

With threats like ransomware and advanced persistent threats (APTs) constantly changing, antivirus software has to keep evolving too. Cloud-based detection and real-time updates now allow antivirus programs to defend against zero-day attacks and protect users from the latest vulnerabilities. Despite all these advances, the battle between antivirus developers and malware creators is ongoing, and it requires constant innovation to stay ahead.

From the Reaper program to today’s AI-powered solutions, antivirus software has evolved a lot, and it’s still on the frontlines of the fight against cyber threats.