Cairo University
Faculty of Computers and Artificial Intelligence

Cairo University, Faculty of Computers and Artificial Intelligence

BCS241 Operating Systems

2nd Semester – 2020 - Comparative Study for Deadlock Handling

Appendix A

Cairo University
Faculty of Computers and Artificial Intelligence

CS342

Operating Systems (2)

2nd Semester 2020 Research

Deadlock Handling Techniques

ID	Name	Email
20170084	Hatem Sayed Ali	hatemgad98@gmail.com
20170136	Atef Magdy Mitwally	atefmagdy12@gmail.com
20170002	Ibrahim Ramadan Abdu	ibrahemramadan130@gmail.com
20170022	Ahmed Sayed Mansour	ahmed.mans20719@gmail.com
20170143	Abdulrahman Bahig Mahmoud	abdelrahmanbahig04@gmail.com

Table of Contents:

List of Figures:

List of Tables:

1. Introduction:

Nowadays operating systems work with multiprogramming and concurrent environments. Mutual exclusion for locking the multi threads actions in system are needed for threads that write in a certain resource. Several threads or as referred to (processes) might require several resources available in the computer hardware, use them and release them for other processes to use.

The life cycle of the process is for :

- **Request :** process request allocation of a certain resource
- **Use and hold :** The process is using the resource currently, so the resource is holden by this process until it finishes.
- **Release :** The process has finished its need for the resource and releases it for other processes to use.

1.1. What is Deadlock

A **circular wait** might happen when a process using a resource while requesting another resource, which is used by a second process, the second process is requesting a resource that is held by a third process etc. This will continue normally until some process in this chain requests the resource held by the first resource. This issue will create a circular wait loop. This problem is known as **Deadlock**.

Figure 1 illustrates the circular wait that leads to deadlock between two processes and two resources.

Process 1 allocates **resource 1** but cannot finish before allocating **resource 2** which is assigned to **process 2**, **process 2** will not release it because it is waiting **process 1** to finish **resource 1** in a waiting circular loop.

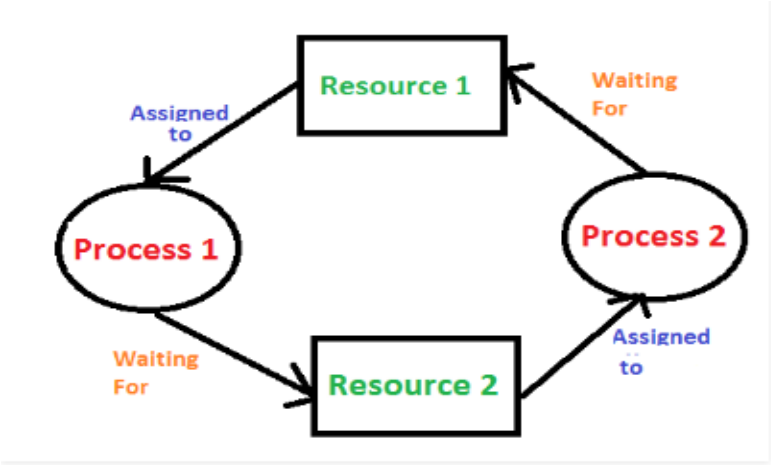

Figure 1. Deadlock between 2 processes and 2 resources

1.2. Deadlock Necessary Conditions:

Deadlock does not happen so frequently due to it must meet certain conditions to occur. These conditions must be simultaneous in the system:

- **Mutual Exclusion:**
  A resource is kept in non-sharable mode (only one process can access at a time).

- **Hold and Wait:**
  The process must be holding one resource (in use) and waiting for another resource hold by other process.
- **No preemption:**
  The resources will not be released except if the process holding voluntarily release it by completing its task.
- **Circular wait:**
  A chain of processes(two or more) waiting for each other to finish to gain each other's resources in a circular form.

We will later discuss how different operating systems handle the deadlock situation and compare between real-time operating systems' deadlock handling.

For applying deadlock handling techniques, **Linux**, **Windows**, **Android**, and **FreeBSD** are selected to be briefly described then their deadlock handling techniques will be discussed.

2. The Operating Systems:

2.1. What is Windows:

Windows is referring to Microsoft Windows, it is a graphical operating system developed and published by Microsoft, designed for both home computing and professional purposes. Windows provide a way to store files, run software, play games, watch videos, and connect to the Internet.

First introduced with version 1.0 on November 10, 1983. Over a dozen versions of Windows were released after that, including the current version, Windows 10. It became most popular in the entire globe and most computers work with Microsoft Windows. It is best known for its user-friendly features and it is easy for users to use.

2.2. What is Linux:

Linux is an opensource operating system that has been around since mid-1990s and has since reached a user-base that spans the globe. It is one of the most popular platforms on the planet, it is everywhere, and many other operating systems are powered by Linux like Android operating system.

Besides being the platform of choice to run desktops, servers, and embedded systems across the globe, Linux is one of the most reliable, secure, and worry-free operating systems available and receives excellent support from a large community of users.

2.3. What is Android:

Android is an open source Operating System based on Linux, made for mobile devices such as smartphones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

Android platform is the most widespread nowadays in the realm of smartphones. It is easy, user-friendly environment and its source code is available under free and open source software license.

2.4. What is FreeBSD

FreeBSD is an operating system derived from BSD the version of UNIX® developed at the University of California, Berkeley. Used to power modern servers, desktops, and embedded platforms. Made specially for platforms which focuses on features, speed, and stability. Its large developing community has continually developed it for more than thirty years. Its advanced networking, security, and storage features have made FreeBSD the platform of choice for many of the busiest web sites and most pervasive embedded networking and storage devices.

FreeBSD offers advanced networking, performance, security and compatibility features missing in other operating systems, even some of the best commercial ones.

3. Deadlock Handling in Different Operating Systems:

3.1. Deadlock Handling Strategies:

There are four main deadlock handling methods used :

- **Ignoring the deadlock**
- **Detecting and recovering from a deadlock**
- **Deadlock prevention**
- **Deadlock avoidance**

3.2. Windows and Linux Operating Systems:

Most of the operating systems including Windows, all its versions and Linux use the same way to handle deadlocks, which is **ignoring deadlocks**.

3.2.1. Ignoring Deadlocks:

A situation in which the system is in a deadlocked state yet there is no way of recognizing what is happening. In this case, the undetected deadlock will cause the system's performance to deteriorate, resources held by waiting processes that cannot finish their task will accumulate, as more resources are being held captive without progressing, more deadlocked process are forming a pile form. Eventually, the system will stop functioning and will need to be restarted manually by the user.

Although this method seems unavailing, it is the common approach in deadlock handling in most operating systems. As expense is considered important, and deadlock in some systems rarely strikes like once or twice per year….

The extra expense in other methods may seem worthless as deadlock is not an every-day problem.

As cheap and low expense is preferred over rare failures. When failure occur, the restarting manually option is the only option available.

3.3. Android Operating System:

Android uses the **detect and recovery technique**, meaning that the system leaves the deadlock to take place, it detects if deadlock happens and recover from it.

3.3.1 Detect Deadlock:

For overcoming the situations of **deadlocks**, the android on a regular basis checks the systems for any deadlocks happening. Resource scheduler can easily help detecting the deadlock by tracking all the resources and processes available in the system.

If deadlocks occur frequently, the operating system should invoke detection process frequently.

Of course, invoking the deadlock-detection algorithm for every resource request will rise a considerable overhead in computation time. Less expensive alternative is simply to invoke the algorithm at defined periods.

For example, force detection once per hour, or whenever CPU utilization drops below a certain level. As the deadlock behavior will eventually cripple the system throughput and causes the CPU utilization to drop.)

If we invoke arbitrary detection algorithm at random points in time, the resource graph may contain many cycles. In this case, deadlocks have already stacked we cannot tell which of the deadlocked processes started the deadlock.

Once a deadlock is detected, recovery from deadlocks will be performed on the deadlocked process and resources by the operating system.

3.3.2. Recovery from Deadlock:

There are two recovery methods that can be used:

- **Abort all deadlocked processes**
  This method obviously will break the deadlock cycle, but at great cost. The deadlocked processes may have made a progress over long time, and the progress of these partial computations will be discarded.
- **Abort deadlocked processes one by one until the deadlock cycle is eliminated**
  This method will add more overhead, since after each process is aborted, the deadlock detecting algorithm must be invoked to determine whether to stop or continue, as more deadlocked processes are to be found.

Android uses the second recovery technique, which will cause the deadlock processes to terminate one by one, with overhead detecting of further deadlocks after each termination.

In the chosen **partially termination method,** the process chosen to be terminated is called **victim**, and the selection itself is called **Minimum cost.**

The recovery will only terminate tasks working, while losing their complete progress but ending the deadlock. Even after ending the processes it will look again for deadlocks, if still exist, it will repeat the termination process until the system becomes deadlock free

3.3.2.1. Minimum Cost factors:

- Priority of the process
- How long the process has computed, and how long will it continue computing
- Type and quantity of resources the process has used
- Whether the resources hold preemptive.
- The process being interactive or batch.
- Number of processes to be terminated.

At last, we preempt the resources held by the victim, abort the process, and restart it in a process called **Roll Back.**

To ensure no starvation will happen to the victim, a process can be selected as a victim only a (small) finite number of times. Thus, before restarting the process the number of rollbacks must be stated.

Terminating a process may lead to UI thread of an Android app being blocked for too long and an "Application Not Responding" (ANR) error is appearing.

If the application is in the foreground state, the system will display a “not responding” dialog to the user, as shown in **figure** 2. The ANR dialog will force the to quit the app.

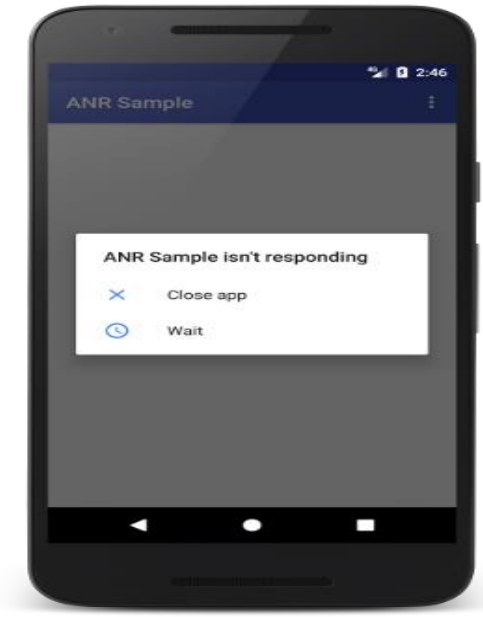

Figure 2. ANR dialog displayed to the user

3.4. FreeBSD Operating System:

FreeBSD has its own deadlock handling technique, which is called **Mutex Control**, it is considered as **Deadlock Prevention strategy**, so first a brief general explanation of the Deadlock Prevention strategy will be shown, then move to how FreeBSD apply its technique.

3.4.1. Deadlock Prevention in General:

As mentioned before, there are four simultaneous conditions for a deadlock to happen, **mutual exclusion**, **hold and wait**, **no preemption** and **circular wait**.

If we succeeded with breaking one of those conditions, we ensure deadlock will never take place in the system, even if the rest of conditions were met. Mutex Control is a technique that lower the chances of deadlocks and other aspects by controlling the mutexes, which leads to less mutual exclusion condition.

3.4.2. Mutex Control:

Mutex Control helps FreeBSD reduce deadlocks but does not totally prohibit it.

Mutex Control implements the violating the mutual exclusion condition, whereas violating any condition can lead to no deadlocks. Some mutexes cannot be preempted, so the probability of deadlock happening is not zero, nevertheless it is extremely reduced.

Mutex control is based on scheduling the processes that demands a certain mutex.

Whenever a process releases its resources, integrate mutex will control the receiving of next processes from a ready queue by performing scheduling algorithms (Round Robin, SJF, FIFO, etc.…). The resources needed by each process will be evaluated to be granted to the ready processes then.

In Figure2, it is shown the implementation of scheduling algorithm for the assessment step in a process requesting a critical section, a mutex is assigned to the process.

Mutex Control leads to having better management and resource allocation, avoiding problems like our major concern the deadlocks, among other problems like (CPU timeouts, deadlocks, interlocking and other aspects).

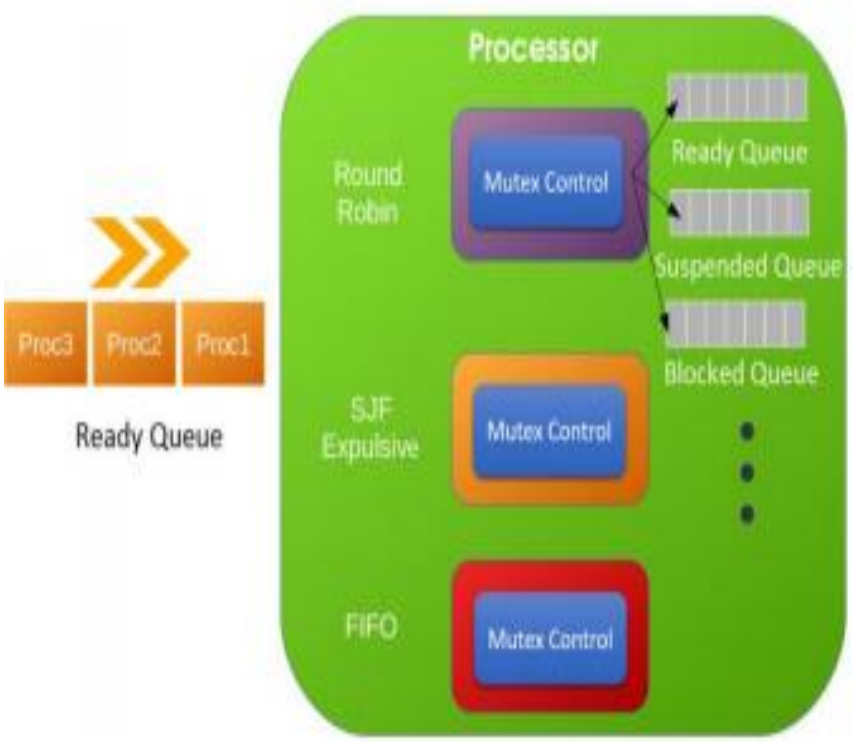

Figure 3. Mutex control and scheduling Algorithms

4. Comparative Overview of the Selected Operating Systems:

To summarize the techniques covered above, we will construct a comparative study between the different operating systems we used to apply the deadlock handling.

For comparing between these operating systems, we will stick to some comparing items:

- **Deadlock rate**
  Indicates the rate which will get affected by the deadlock handling technique.
- **Overhead**
  Overhead of the method used in handling the deadlocks.
- **Efficiency**
  Preview of the system, whether the method used improve the performance of the system or weaken it (deadlock handling and the expenses for handling).
- **Process losing its progress**
  How the deadlock can affect the progress in tasks and the loose of data.

4.1. Table of Comparison:

Operating System	Windows	Linux	FreeBSD	Android
Deadlock Handling Technique	-Ignoring the deadlock	-Ignoring the deadlock	-Mutex Control (Deadlock Prevention)	-Detect and Recover
Deadlock Handling	-Highest	-Highest	-Low	-High -Will recover from deadlock state
Overhead	-Low	-Low	-High	Highest
System Performance	-Highest	- Highest	-Good -Overhead of scheduling processes and mutexes	-Low -Costly -Deadlocks solved
Process losing its progress	-Very likely to happen. -User might have to restart manually, and all tasks opened will be lost	-Very likely to happen. -User might have to restart manually, and all tasks opened will be lost	-Low rate -If it happens user will have to restart manually, and all tasks opened will be lost	-Medium rate Might lose important data, but most likely tasks will process again

Table 2 Comparative table

4.2. Overview of comparative Features:

In this section we will discuss deeply the operating systems.

4.2.1. Deadlock Handling Comparison:

If we only consider the deadlock situation without concerns about the system performance:

-Android exploits the best deadlock handling technique in comparison with the rest(deadlock happens then get detected and recovered) still can have its own drops like the frequency of deadlock detection fails to match with deadlock timing

- FreeBSD comes after with a low rate of deadlocks occurrences.

-Windows and Linux are the worst in handling deadlocks.

4.2.2. Overhead Expenses:

- Windows and Linux have no extra overhead and low cost, as nothing is done.

- FreeBSD comes next with scheduling algorithms at the critical section.

- Android gives the biggest overhead, with scheduled detection overhead, and recovery expenses.

4.2.3. Whole System Performance:

- FreeBSD may be the best, not because it handles deadlocks only, but its mutex control maintains better management and resource, allocation to avoid problems CPU timeouts, deadlocks, interlocking and other aspects.

- Windows and Linux comes second, no overhead of detecting deadlocks, no expenses just deadlocks popping up every now and then.

- Android will have the less performance due to its detecting algorithm that works on regular basis.

4.2.4. Processes Progress Loosing:

-Android might restart its deadlocked process, but most times it will be done with the delay of restarting from over again, the process will restart from the very beginning, not from a state as saving state is a very expense operation.

- FreeBSD comes next with a very low probability to have deadlock, but if happens, all progress by deadlocked processes will be lost due to the urge to restart the device manually.

- Windows and Linux are most likely to get hit by deadlocks and eventually the system will collapse in a pile of further deadlocks until end-user restarts manually. All progress of process will be lost without a restart or recover by system.

5. Conclusion:

With all the covered deadlock handling techniques and operating systems, we might conclude it is better to seek the needs of your system, no such technique is the best fit for any operating system, but as end-user computers can be tolerate towards deadlocks.

However, in this comparative study showed that ignoring is the best performance, as deadlock is not a frequent struggle, and its faults might be tolerated.

In general, researchers have argued that none of the basic approaches mentioned above alone is appropriate for the entire spectrum of resource-allocation problems in operating systems. Resource allocating may differ in the same operating system, The basic approaches can be combined allowing us to select an optimal approach for each class of resources in a system.

6. References:

- *Silberschatz, A. Galvin, P. Gagne, G. 2013. Operating System Concepts – 9th Edition: Courier Kendallville. United States. 920.*

- *Acosta, L. Acosta, C. García, N. Villalobos, O. 2014. Design of a Mutual Exclusion and Deadlock Algorithm in PCBSD – FreeBSD. Universidad Distrital Francisco José de Caldas, Bogotá, Colombia.*

- *White, S. 2019. Deadlock: What it is and its condition. [https://www.allassignmenthelp.com/blog/deadlock-what-it-is-and-its-condition/]. Accessed May 16, 2020.*

- *JavaTpoint. 2011. OS Tutorial.[https://www.javatpoint.com/os-strategies-for-handling-deadlock]. Accessed May 16, 2020.*

- *Guru99. 2020. Operating System Tutorial. Introduction to DEADLOCK in Operating System. [https://www.guru99.com/deadlock-in-operating-system.html#1]. Accessed May 16, 2020*

- *Pipalva, V. Benjamin, E. 2014. Introduction to Android OS. 13 pp.* [https://prezi.com/pwxtdbdndbym/introduction-of-android-os/]

- *Android Developers powered by Google. 2019. ANRs. [https://developer.android.com/topic/performance/vitals/anr#deadlocks]. Accessed May 17, 2020.*

- *FreeBSD. The FreeBSD Project. [https://www.freebsd.org/]. Accessed May 18, 2020.*

- *OpenSource.com supported by Red Hat. 2019. What is Linux. [https://opensource.com/resources/linux]. Accessed May 18, 2020.*

- *The Linux Foundation. 2020 . What Is Linux. [https://www.linux.com/what-is-linux/].Accessed May 18, 2020.*

- *Computer Hope. 2020. Windows. [https://www.computerhope.com/jargon/w/windows.htm]. Accessed May 18, 2020.*