Important algorithms of UNIX operating system

Important algorithms of UNIX operating system

Algorithm : getblk

Input : 1. File system number

2. Block number

Output : Locked buffer which can be used in any algorithm.

Description :

- This algorithm is used to allocate requested buffer from buffer pool.
- In buffer pull all buffers are maintained in two data structures as free list of buffers and used list of buffer.
- If the requested buffer is already present in buffer pool then we get that buffer otherwise we have to use some other empty buffer.

Algorithm : brelease

Input : 1. Locked buffer

Output : Nothing

Description :

- This algorithm is used to release the buffer after it's used.
- We have to wake up all the processes which are waiting for any buffer to become free.
- Then we have to wake up all processes which are waiting for this specific buffer to become free.
- If the contents of buffer are valid we have to insert the buffer at end of free list.
- Otherwise insert the buffer at start of free list.

Algorithm : breada

Input : 1. Block number to read immediately (first block)

2. Block number to read asynchronously (second block)

Output : Buffer containing data for first block

Description :

- This algorithm is used to read the contents from hard disk to system buffer which is returned by getblk() algorithm.
- There are four scenarios in this algorithm :
  1. First and second buffer is in buffer pool – In this case we have to return first buffer immediately.
  2. First and second buffer not in buffer pool – In this case we have to start reading the contents for first buffer and start reading for second buffer asynchronously.
     We have to wait until first buffer contains valid data.
  3. First buffer is in buffer pool and second buffer is not in buffer pool – In this case we he to start reading second buffer asynchronously and return first buffer.
  4. First buffer not in buffer pool and second buffer is in buffer pool – In this case we he to start reading first buffer and wait until first buffer contains valid data.

Algorithm : bread

Input : 1.The required block

Output : Buffer containing valid data

Description :

- This algorithm get the buffer by calling the getblk() algorithm.
- If the buffer returned by getblk() already contains the valid data then return that buffer as it is.
- Otherwise we have to read the contains of that block from hard disk and wait till that buffer contains valid data.

Algorithm : bwrite

Input : 1. The buffer that we wanted to write.

Output : Nothing

Description :

- This algorithm is used to write the contents into the harddisk.
- For that purpose we have to initiate hardware controller and wait until contents are written on harddisk.
- If buffer is marked as delayed write then insert that buffer at beginning of free list.

Algorithm : iget

Input : 1. File system number

2. Inode number

Output : Locked inode.

Description :

- When we open existing file this algorithm gets internally called.
- This algorithm is similar to getblk() algorithm.
- We the specified inode is present in inode pool then that inode is returned directly.
- Otherwise one free inode gets selected from free list and then read the contents of that inode from hard disk.
- This algorithm increases the reference count of an inode.

Algorithm : iput

Input : 1. Pointer to an incore inode

Output : Nothing

Description :

- When we close an opened file this algorithm gets internally called.
- Main task of this algorithm is to decrement the reference count.
- If after decrementing the reference count it becomes zero then that inode become part of free list of inodes.
- If the contents of inode get modified then we have to update those contents on hard disk inode.
- If link count of an file is also zero then free all the blocks which are allocated for that file and set file type of that file as zero also call algorithm ifree().

Algorithm : bmap

Input :
1. Inode of the file
2. Byte offset of the file

Output :
1. Physical disk block number corresponding to file
2. Byte offset into block
3. Bytes of I/O in block
4. Read-ahead block number

Description :

- This algorithm accepts inode and offset and returns the physical block number in which this offset resides.
- It returns the offset from that block in which we are interested.
- It also returns how many bytes we can read from that block and check whether read ahead operation is applicable or not.
- This algorithm decides whether to call breada() or bread() algorithm.

Algorithm : namei

Input :
1. Path name string

Output : Locked inode of desired file.

Description :

- This algorithm is used to retrieve the inode from specified path.
- First check whether path is absolute path or relative path to decide the start of inode search.
- If path is absolute path then start search from root directory otherwise start from the current directory.
- Read every component from the path and match that component in directory file and if component matches retrieve its specified inode and follow same procedure till path we get desired inode.
- This algorithm internally uses bmap(), bread(), brelease(), iput() and iget() algorithms.

Algorithm : ialloc

Input : 1. File system

Output : Locked inode

Description :

- When we want to create new file this algorithm gets called.
- As information about free list of inode is maintained inside the superblock we have to wait till super block become unlock.
- Get the inode number from superblock and pass that inode number to iget().
- If free list of inodes from super block is empty we have to fill that list by taking help of remembered inode number.
- Initialize that inode and update it immediately in hard disk.

Algorithm : ifree

Input : 1. Inode number to free

Output : Nothing

Description :

- When we delete any file this algorithm gets internally called.
- The task of this algorithm is to insert the inode number in free list of inodes in superblock.
- If superblock is locked we have to return immediately because sooner or later this inode gets considered as free inode.
- If free list of inode is full then we have to check our inode number with remembered inode otherwise insert inode in free list of inodes.
- If our inode number is less than the remembered inode then our inode become new remembered inode.

Algorithm : alloc

Input : 1. File system number

Output : Buffer for new block

Description :

- If want to allocate new block for file then this algorithm gets internally called.
- As information about free list of blocks is maintained inside the superblock we have to wait till super block become unlock.
- Remove the block number from free list of blocks and if this is the lst block from the list then fill the list again with its contents.
- Get the buffer for specified block by getblk() and zero out its contents.

Algorithm : free

Input : 1. Block number that we want to free.

Output : Nothing

Description :

- When we delete contents of file its associated block gets freed by this algorithm
- Insert that block number in free list of blocks.
- If list of free blocks is full then copy that list in our block then insert block number.

Algorithm : open

Input : 1. File name string (with absolute/relative path)

2. Type of open (like for reading or writing or both)

3. File permissions (for creating the file)

Output : File descriptor for specified file

Description :

- This algorithm is used to open an existing file or to create new file.
- First we have to get inode from specified path by using namei().
- After namei inode gets placed in Incode inode table.
- Create file table entry and initialize its contents as offset, capability, count.

- Get first unused entry from User File Descriptor Table and insert address File Table entry.
- If file is opened in truncate mode then free all its blocks by free().

Algorithm : close

Input : 1. File descriptor

Output : Nothing

Description :

- This algorithm is used to close an opened file.
- Drop the count field from File table by one.
- After decrementing it becomes zero then remove that entry.
- Drop the reference count of inode from Incode Inode Table, as if it becomes zero then it will become part of free list of inode.
- Free the entry of User File Descriptor Table entry by inserting NULL.

Algorithm : read

Input : 1. User file descriptor

2. Address of a buffer (local to process)

3. Number of bytes to read

Output : Number of bytes actually read

Description :

- This algorithm is used to read contents of regular file into local buffer.
- From file descriptor we have to access its file table entry to check the capability of opened file.
- Set internal I/O parameters in UAREA as user address, byte count, file offset, flag etc.
- Get the current offset from file table and pass that offset to bmap().
- Read the block by using bread or breada.
- Copy the contents from system buffer into local buffer.
- Update file table offset entry after reading gets completed.

Algorithm : write

Input : 1. User file descriptor

2. Address of a buffer (local to process)

3. Number of bytes to write

Output : Number of bytes actually write

Description :

- This algorithm is used to write contents in regular file from local buffer.
- From file descriptor we have to access its file table entry to check the capability of opened file.
- Set internal I/O parameters in UAREA as user address, byte count, file offset, flag etc.
- Get the current offset from file table and pass that offset to bmap().
- Copy the contents from local buffer into system buffer.
- Write the block by using bwrite().
- Update file table offset entry after reading gets completed.

Algorithm : lseek

Input : 1. File Descriptor

2. Offset

3. Reference

Output : New file offset

Description :

- By using this algorithm we can perform random access in regular file.
- Access files table entry from file descriptor.
- If reference is zero then set input offset in offset entry of file table.
- If reference is two then set input offset plus size of file in offset entry of file table.
- If reference is one then add input offset in offset field of file table.

- After calling lseek function next reading and writing is starting from this offset.

Algorithm : creat

Input : 1. File name

2. File permissions

Output : File descriptor

Description :

- This algorithm is used to create new regular file.
- If inode of specified file by namei().
- If file is already existing then free all blocks of that file by free().
- Otherwise assign new inode by ialloc() and create directory entry of that file in its parent directory.
- Create file table entry and return file descriptor from UFDT.

Algorithm : mknod

Input : 1. Path name

2. Type & permissions

3. Major & minor numbers for device file if any

Output : Nothing

Description :

- This algorithm is used to create special files as device file, named pipe, directory file.
- If we want to create device file or directory file then caller of this algorithm must be super user.
- Allocate new inode for file by using ialloc().
- Create entry in parent directory and if name already exists then return error.
- If new file is device file then insert its major number and minor number.

Algorithm : chdir

Input : 1. Pathname string

Output : Nothing

Description :

This algorithm is used to change current directory of process.

Get the inode of new current directory by namei().

Release inode of existing current directory by iput().

Place that inode number in UAREA.

After calling chdir searching path of relative path gets changed.

Algorithm : chroot

Input : 1. Pathname string

Output : Nothing

Description :

This algorithm is used to change current root of process.

Get the inode of new current root by namei().

Release inode of existing current root by iput().

Place that inode number in UAREA.

After calling chroot searching path of absolute path gets changed.

Algorithm : pipe

Input : 1. Pointer to an integer array with empty elements

Output : Pointer to an integer array, elements of which are now valid file descriptors, one for reading and

other for writing (when they are parameters, they were.

0^{th} element – Reader, 1^{st} element – Writer.

Description :

- This algorithm is used to create unnamed pipe which is used for IPC.
- This pipe can be accessed by related processes only.
- New inode gets assign by alloc() from pipe device.
- Two file table entries gets created one for reader process and one for writer process.
- Tow UFDT entries get allocated and fill those entries in out parameter.
- Initialize the extra elements of pipe inode as
  - Byte offset in pipe
  - Reader count, for reader processes.
  - Writer count, for writer processes.
  - Read pointer & Write pointer

Algorithm : dup

Input : 1. File descriptor.

Output : New file descriptor.

Description :

- This algorithm is used to duplicate an existing file descriptor.
- From the file descriptor access the File table entry and increase the count from file table.
- Search first unused entry from UFDT and insert address of that file table entry.

Algorithm : dup2

Input : 1. Existing File descriptor.

2. New File descriptor.

Output : New file descriptor.

Description :

- This algorithm is used to duplicate an existing file descriptor.
- From the file descriptor access the File table entry and increase the count from file table.
- If file of second file descriptors file is already opened then close that file and use its entry.
- If both the parameters are same then return input file descriptor

---

Algorithm : mount

Input : 1. File name of Block Special file

2. Directory name of "mounted on"

3. Options (read - only).

Output : Nothing

Description :

- This algorithm is used to mount an file system on existing file system.
- Caller of this algorithm must be an super user.
- Get the inode of block special file and mounted on directory by namei().
- Find empty slot in mount table and initialize the contents of mount table as
  - Device number.
  - Pointer of inode of "mounted on" directory.
  - Pointer of inode of root of "mounted file" system.
  - Pointer to buffer of super block of mounted file system.

Algorithm : umount

Input : 1. Special file name of file system to be unmounted

Output : Nothing

Description :

- Get inode of special file name by namei() and from that inode extract major and minor number.
- By using major number and minor number search the entry from mount table.
- If files from unmountable file system are still in use then we cannot unmounts the file system.
- Release the inode of mount point and free the slot from mount table.

Algorithm : link

Input : 1. Existing file name

2. New file name

Output : Nothing

Description :

- This algorithm is used to create hard links.
- First get the inode of existing file by namei().
- Increase the link count of the existing files inode.
- Get parent directory's inode of new path by namei and insert new name in that directory and inode number of existing file.

Algorithm : unlink

Input : 1. pathname

Output : Nothing

Description :

- This algorithm is used to delete an existing file.
- Get inode of parent directory by namei().
- Search file name from that directory and write inode number as zero.
- Drop the link count of that fie by one and if it become zero then free the blocks by free() and free the inode by ifree().

Algorithm : int_hand

Input : Nothing

Output : Nothing

Description :

- This algorithm is used to handle the interrupt.
- Before handling the interrupt save the context layer.
- Find source of interrupt and open interrupt vector table (IVT).
- Invoke the interrupt handler and restore the previous context layer.

Algorithm : syscall

Input : system call number

Output : Result of system call

Description :

- System call number is retrieved from the trap instruction.
- Search system calls entry from system call table and determine the parameters required for system call.
- Get the address of system call from system call table and invoke that code from kernel address space.
- If there is as error while invoking the system call then turn on the carry bit and set error code in appropriate register.
- If system call is successful then copy its return value in register 0 &1.

Algorithm : sleep

Input : 1. Sleep address

2. Sleep priority.

Output : 1. If process's sleep is set to catch the signals (for its wake up), then the value returned is 1
2. or when long jump algorithm is executed as return status;
3. or 0 in all other cases

Description :

- This algorithm change state of process to sleep state (state 4).

- Every process goes into sleep state due to some event.
- When process goes into sleep state it placed into sleep hash queue.
- Save the sleep address in process table.
- If sleep of process is not interruptible then does the context switch and return 0 from algorithm.
- If sleep of process is interruptible then does the context switch and check whether signal arrived or not.
- If there is a signal then return 1 from algorithm

Algorithm : wakeup

Input : 1. Sleep address

Output : Nothing

Description :

- This algorithm is used to switch state of the process from sleep to ready to run.
- By using sleep address remove all processes from sleep hash queue which are sleeping on that address.
- Change state of process as ready to run.
- If this process is not yet loaded in memory then wake up swapper process to swap in this process.

Algorithm : fork

Input : Nothing

Output : Nothing

Description :

- This algorithm is used to create new light weight process.
- Allocate new PID and process table slot for new process.
- Mark state of this new process as being created (state 9).
- Copy the process table slot of parent process to child process.
- Create copy of parent's text, data, and stack.
- If parent process is running then return PID of child process and if child is running then return 0.

Algorithm : issig

Input : Nothing

Output : true – if process received a signal that is not ignored.

false – otherwise(i.e. either signal are ignored or there are no signals)

Description :

- This algorithm is used to check whether signal is arrived or not.
- This algorithm gets called from process go to sleep state from kernel mode and when process go to kernel mode from ready to run state.
- Information about the pending signal is maintained inside process table slot.
- If there is a death of child signal for the process and we want to handle that signal then return true.
- Otherwise remove its process table entry and return false.
- If want to ignore other signal then turn off the signal bit and return false otherwise return true.

Algorithm : psig

Input : Nothing

Output : Nothing

Description :

- This algorithm is used to handle the signal.
- For any signal we can perform any of this three actions as
  - Handle the signal
  - Ignore the signal
  - Exit on arrival of signal
- Before taking any action on signal turn off the signal bit from process table.
- UAREA of a process maintain information of the signal and its expected action.

- If want to handle the function then get the address of that user defined function and invoke that function.
- If want to exit on arrival of signal then call exit() algorithm.

Algorithm : signal

Input : 1. Signal number

2. Action on that signal

Output : Address of old registered function.

Description :

- This function is used to decide the action on arrival of particular signal.
- First parameter is the signal number for which we have to decide the action.
- If want to handle the signal then second parameter is address of function which is considered as signal catcher function.
- If want to ignore the signal then second parameter is 1 and if want to exit on the arrival of signal then second parameter is 0.
- When we call these function contents of UAREA gets updated which are concern with signal.

Algorithm : kill

Input : 1. PID of process

2. Signal number

Output : Nothing

Description :

- This function is used to send signal to particular process which is mentioned as a second parameter.
- If pid is +ve integer, kernel send the respective signal to the process having its process ID as pid.
- If pid is 0, then kernel sends respective signal to all processes in the sender's process's group.

- If pid is –1, then kernel sends respective signal to all processes whose "REAL USER ID" is equal to the "effective user id" of the sender.
- If pid is –ve value, but not –1, then kernel sends respective signal to all processes in that group which is equal to the absolute value of pid.

Algorithm : exit

Input : 1. Exit status return to parent

Output : Nothing

Description :

- This algorithm gets directly or indirectly gets called when our process terminates normally or abnormally.
- This algorithm internally invokes all exit handlers which are registered by that exited process.
- As our process is going to be exited we have to ignore all signals which are not yet handled.
- We have to release the inode of current directory and current root by iput.
- We have to remove all page tables which are associated with the exited process.
- Maintain the information of this process in global account file.
- Parent process of all children's of this exited process becomes process 1.
- Send death of child signal to parent process.
- As our process is going to be exited we have to perform context switch to schedule some new process.

Algorithm : wait

Input : 1. Address of integer variable to store status exiting process.

Output : Child PID.

Description :

- This algorithm is used to synchronize the parent's executions with the termination of child processes.
- When we call this system call our process waits till receiving the death of child signal of child process.
- As this algorithm is related to death of child signal due to which final behavior of this algorithm is depend upon the action taken on that signal.
- These algorithms wait till that process has zombie child and if zombie child gets detected then we have to free its process table entry and add its CPU usage to the parent CPU usage.
- If parent process has no children then we have to return from algorithm.
- If we are going to ignore the death of child signal then our process wait till all of its child process terminates.
- But if we want to handle death of child signal or exit on arrival of that signal then our process continues its execution after receiving first death of child signal of child process.
- Means if our process wants to wait till termination of all of its child then we have to ignore death of child signal by calling signal() system call.

Algorithm : exec

Input :

1. Executable file's name (with path).
2. Parameter list to executable program.
3. Environment variable list.

Output : Nothing

Description :

- This algorithm is used to execute new process in address space which is created by fork system call.
- As executable file is present is in hard disk we have to read that file check whether it is an executable file or not.
- In executable file headers there is information about sections which are present in file.
- First we have to detach all regions attached to our process by fork system call by calling detachreg().

- Now new regions from that executable file get attached to our address space.
- After this new process start executing which is considered as a full blown process.

Algorithm : xalloc

Input : 1. Inode of executable program file.

Output : Nothing

Description :

- This algorithm is used to allocate memory for text region.
- As we are never going to change the contents of text region there is no need to allocate separate text region for every process whiose text region is already loaded in RAM.
- In this algorithm kernel first searches the existing text region in memory and if that region is available in memory then we have to attached that already loaded region.
- Otherwise allocate new text region.

Algorithm : brk

Input : 1. New break value (Virtual address)

Output : Old break value.

Description :

- This algorithm is used to increase or decrease size of data region.
- We cannot change size of text region and stack regions size gets changed by kernel whenever required.
- This algorithm accepts the new virtual address upto which we have to allocate the memory.
- Before allocating physical memory, memory manager has to check whether increasing the size of region is permissible or not.
- If it is possible to increase size of region then internally growreg algorithm gets called.
- This algorithm returns the virtual address from where the memory gets allocated.

Algorithm : sbrk

Input : 1. Number of bytes to allocate.

Output : Old break value.

Description :

- This algorithm is exactly same as brk algorithm but instead of accepting the virtual address it accept the number of bytes that we want to allocate.
- If parameter is positive then memory gets allocated and if it is negative then memory gets de allocated.
- This algorithm internally calls growreg() algorithm to physically allocating the memory.
- This algorithm gets internally called by malloc(), calloc(), free(), realloc() library functions.

Algorithm : start

Input : Nothing

Output : Nothing

Description :

- This algorithm is used to create first process of process subsystem i.e. process 0.
- This algorithm first creates and initializes the data structures of kernel.
- It creates hard coded environment for process 0 as its page table, UAREA, Process table slot etc.
- This process 0 is responsible for creating new process 1 by calling fork system call.
- For process 1 it allocates regions which are required and call exec() algorithm to execute process from /etc/init.
- After completing the above task process 0 becomes swapper process by calling swapper algorithm.

Algorithm : init

Input : Nothing

Output : Nothing

Description :

- This algorithm is for process 1.
- This process opens file from /etc/inittab which contains names of the processes that init process has to execute.
- This algorithm executes every process from inittab according to the state i.e.single user or multi user.
- After executing all process we have to wait till termination all process.
- If action of process is mentioned as re-spawn then we have to execute that process again.

Algorithm : schedule_process

Input : Nothing

Output : Nothing

Description :

- This is a scheduling algorithm which is used to process from ready to run state.
- This algorithm schedule process depends on its scheduling priority.
- It selects highest priority from ready to run state.
- Now kernel has to switch context to that selected process.
- If there is no process to schedule then machine goes into idle state and it come out of idle state after getting clock interrupt.

Algorithm : malloc

Input : 1. Map address

2. Requested number of free blocks.

Output : Address

Description :

- This algorithm is used to allocate space in swap partition.
- To maintain information about free memory from swap partition map data structure is maintained which contains starting block number and number of blocks free after that block number.
- This algorithm checks every entry from map till requested number of blocks is available.
- If our request is exactly matches with the free blocks from map entry then that entry gets removed otherwise we have to adjust that entry.