QUESTION 1

System Call Tracing Implementation

This modification facilitates the printing of system calls, including relevant information such as process id, process name, and return value. The tracing process initiates upon entering the 'trace' command and continues until the 'untrace' command is executed.

proc.h:

Declare a trace function on line 52.

proc.c:

Include the trace header file at line 9. Initialize the trace status in the allocproc function at line 91. Implement trace logic inside the fork function at line 204.

syscall.h:

Define the sys trace command at line 23..

syscall.c:

Change the syscall function to define the logic of the trace syscall and how it prints.

usys.s:

Declare the trace syscall at line 32.

user.h:

Declare the trace syscall at line 26.

trace.h:

Define values of constants used for syscall tracing. Create a structure for the trace function.

trd.c:

Tracing the parent, forked and then untracing in case UNTRACE flag is set

Sh.c

All about starting the tracing when “trace” is written and stop when “untrace” is written

System Call Tracing

To initiate tracing, use the 'trace' command. This will begin tracing system calls for every subsequent command. To halt tracing, use the 'untrace' command.

Ensure that you implement the details for the trace function and the sys_trace function based on your specific requirements. Also, choose an unused syscall number for SYS_trace.

QUESTION 2

Process Information Printing Implementation
This set of changes enhances the system by allowing the printing of information about a process.

proc.c:
Defines the logic for printing process information.

syscall.h:
Defines the 'ps' syscall at line 24.

usys.s:
Declares the 'ps' syscall at line 33.

defs.h:
Declares the ps function for the kernel syscall suite at line 126.

sysproc.c:
Returns the result of the 'ps' syscall (implemented in sys_ps) at line 105.

user.h:
Defines the 'ps' syscall at line 27.

syscall.c:
Adds the 'ps' syscall to the syscall array.

Process Information Printing

This modification allows you to retrieve and print information about a process using the 'ps' syscall. Ensure that you handle the implementation details of the 'ps' function in the kernel suite and appropriately define the 'sys_ps' function.

This new functionality can be utilized by invoking the 'ps' syscall as part of your system calls.

QUESTION 3

Command History and Arrow Navigation Implementation
This set of changes introduces a command history feature that stores the history of the past 16 commands (maximum) and enables the usage of the up and down arrow keys for navigation.

console.c:

Implements the getcmdfromhistory function that is called by sys_history to write the requested command history into the buffer.
Adds logic for navigating up and down to retrieve previous and next commands in the console_intr function at line 373.
Manages the command history storage and retrieval in the commands_in_history function at line 430.
Implements the save_command_in_history function at line 507.
Defines a structure for storing the buffer array, length of each command string, index of the last entered command, and current history view at line 149.

defs.h:

Declares the getcmdfromhistory function for the kernel-side syscall at line 25.

syscall.h:

Defines the 'history' syscall at line 24.

syscall.c:

Adds the 'history' syscall to the syscall array.

user.h:

Defines the 'history' syscall at line 28.

usys.s:
Declares the 'history' syscall at line 33.

sysproc.c:
Implements the sys_history function, which returns the result of getcmdfromhistory.
Command History and Arrow Navigation
This enhancement allows users to access a history of the past 16 commands and use the up and down arrow keys to navigate through command history. The 'history' syscall facilitates the retrieval of command history.

Ensure you handle the implementation details of the getcmdfromhistory function, and appropriately define the sys_history function. This feature enhances command line interaction by providing a convenient way to review and reuse past commands.

QUESTION 4

Extended Wait System Call (wait2) Implementation
This set of changes extends the wait system call to wait2, providing additional information about a child process, including its creation time, sleeping time, running time, and ready time. These statistics are stored in the proc structure.

proc.h:
Extend the proc struct by adding the following fields:

ctime: Creation time of the process.
stime: Sleeping time of the process.
retime: Ready time of the process.
rutime: Running time of the process.
Add an updateStats() function responsible for updating these fields.

proc.c:
In the allocproc function, initialize ctime to the current number of clock ticks (ticks).
Initialize stime to 0, retime to 0, and rutime to 0.

Implement the updateStats() function. Do not update on ZOMBIE state.

Implement the wait2 function, which is used by the kernel-side system call SYS_wait2 to retrieve the required statistics.

defs.h:

Include the prototype of the wait2 function for the kernel-side system call.

sysproc.c:

Implement the kernel-side system call sys_wait2, which calls the wait2 function defined in proc.c.

syscall.h:

Assign the number 25 for the SYS_wait2 system call.

syscall.c:

Add a pointer to the externally defined sys_wait2 in the array of system calls at index 25.

usys.S:

Add the line SYSCALL(wait2), which stores the value of the system call defined in syscall.h in some processor register. This will be used in syscall.c to find the index in the array of function pointers of system calls.

user.h:

Add a function prototype for the user-side wait2 system call.

trap.c:

Update stats with the timer at line 54.

wait2.c:

Extended Wait System Call (wait2)
This enhancement provides extended information about a child process, including creation time, sleeping time, running time, and ready time. The wait2 system call retrieves these statistics for a child process. Ensure proper implementation details for the added fields and functions, handling ZOMBIE state appropriately. The user can now access more detailed information about the execution history of child processes.