Stage 20

Fork system call

- Heap and code shared
- User stack is copied where as kernel is not
- Returns to parent and return value is PID of child
- Child in created state and return value is 0
- Where does ExpL allocate memory for variables of user defined data type?
    - Variables of user defined data type are allocated memory in stack, same as any variable of primitive data type. Every variable in ExpL is allocated one word memory in stack. Variable of primitive data type saves actual data in the word that is allocated to it, but variables of user defined data type stores the starting address of the object. Alloc() library function allocates memory for an object in heap and returns the starting address. This return address is stored in the stack for corresponding variable.
- Since the Parent and child processes can concurrently access/modify the heap pages, they need support from the OS to synchronize access to the shared heap memory.
- No sync needed for code and heap since no write access.

What fork does

- change mode flag, switch kernel stack
- heap page allocation.
- new stack and user area page.
- Initialization of the process table for the child process
- UPTR field should also be copied from the parent process
- MODE FLAG(since user mode), KPTR(since user mode) and TICK fields of the child process to 0
- PID of the parent is stored in the PPID field of the process table of the child.
- **The per-process resource table has details about the open instances of the files and the semaphores currently acquired by the process.** We copy per-process table of parent to child.
- Copy the per-process disk map tableof the parent to the child. This will ensure that the disk block numbers of the code pages of the parent process are copied to the child.
- BP = [SP]
- Set up return values in the user stacks of the parent and the child processes. Store the PID of the child process as return value to the parent and 0 as the return values to the child. Reset the MODE FLAG of the parent process.

Get pcb

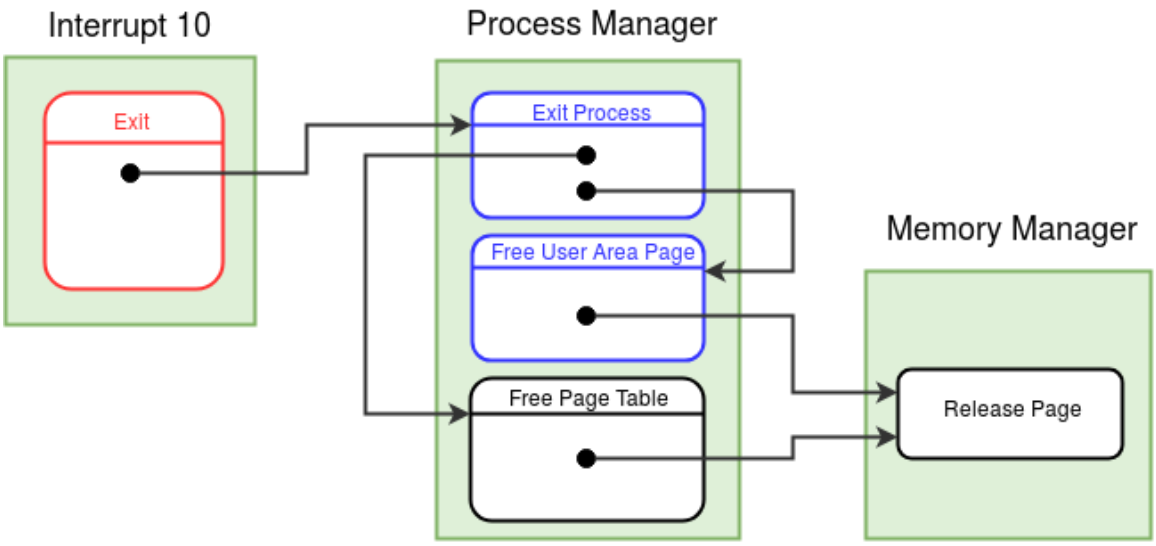
- Finds out a free process table entry and returns the index of it to the caller.
- maximum 16 processes can run simultaneously
- If a free process table is not available, returns -1

- In such case, store -1 as the return value in the stack, reset the MODE FLAG (to 0), switch to user stack and return to the user mode from the fork system call. When PID is available, proceed with the fork system call.
- **Set the STATE to ALLOCATED => Process Table entry has been allocated for the process, but process creation is not complete**

Scheduler

- BP = [SP]

Exit system call



MISC

- Fork system call returns to the parent process. IRET sets the value of IP register to the return address at the top of the stack, pointed to by the SP register. The machine translates the logical SP to physical SP using the page table pointed to by the PTBR register, which points to the page table of the parent. Subsequent instruction fetch cycles continue to proceed by translating the value of IP using the PTBR value, which points to the page table of the parent process. The parent process continues execution till a context switch occurs.
- The fork system call which invokes the Get Pcb Entry function for the child process might block before completing process creation (if the Get Free Page function finds no free page in memory and invokes the scheduler). In such case, the scheduler must not try to run the new process as its creation is not complete.

Process Table entries

- **TICK** (1 word)- keeps track of how long the process was in memory/ swapped state. It has an initial value of 0 and is updated whenever the scheduler is called. TICK is reset to 0 when a process is swapped out or in.
- **PID** (1 word) - process descriptor, a number that is unique to each process. This field is set by Fork System Call. In the present version of eXpOS, the pid is set to the index of the entry in the process table.
- **PPID** (1 word) - process descriptor of the parent process. This field is set by Fork System Call. PPID of a process is set to -1 when it's parent process exits. A process whose parent has exited is called an Orphan Process.

- **USERID** (1 word) - Userid of the currently logged in user. This field is set by Fork System Call.
- STATE (2 words) - a two tuple that describes the current state of the process. The details of the states are explained below.
- **SWAP FLAG** (1 word) - Indicates if the process is swapped (1) or not (0). The process is said to be swapped if any of its user stack pages or its kernel stack page is swapped out.
- **INODE INDEX** (1 word)- Pointer to the Inode entry of the executable file, which was loaded into the process's address space.
- **INPUT BUFFER** (1 word) - Buffer used to store the input read from the terminal. Whenever a word is read from the terminal, Terminal Interrupt Handler will store the word into this buffer.
- **MODE FLAG** (1 word) - Used to store the system call number if the process is executing inside a system call. It is set to -1 when the process is executing the exception handler. The value is set to 0 otherwise.
- **USER AREA SWAP STATUS** (1 word) - Indicates whether the user area of the process has been swapped (1) or not (0).
- **USER AREA PAGE NUMBER** (1 word) - Page number allocated for the user area of the process.
- **KERNEL STACK POINTER** (1 word) - Pointer to the top of the kernel stack of the process. The **offset of this address within the user area** is stored in this field.
- **USER STACK POINTER** (1 word) - Logical address of the top of the user stack of the process. This is used when the process is running in kernel mode and the machine's stack pointer is pointing to the top of the kernel stack.
- **PTBR** (1 word) - pointer to PER-PROCESS PAGE TABLE
- **PTLR** (1 word) - PAGE TABLE LENGTH REGISTER of the process.