Group 28:

A;
This system call returns the parent processes' pid. In xv6, all processes but init have a parent, so we implement a check that if pid is 1, then we return ppid as -1.
For the general case, we acquire the wait_lock, get access to the parent process' process structure, and access it's pid.
B:
We simply call the yield() function implemented in proc.c and return 0.
C:
Given a virtual address A, we return
walkaddr(myproc()->pagetable, A) + (A & (PGSIZE - 1))
D:
This system call is implemented by using the fork() call's template. In the call, the process's trapframe program counter is set to that of the passed function, thus changing control to the function, in the child.
Explanations of forkf() calls:
For the function f returning 0, it prints Parent and Child independently with their respective PIDS.
This is because the forkf() function returns the pid of the forked child to the parent process and the return value of the function passed to forkf is stored in the a0 register of the trapframe, which is returned to the child process.
When f returns 1, then the child process also gets 1 and so both the parent and child process enter the x>0 if statement, causing both the processes to print Parent.
When f returns -1, then the child process enters the x<0 block and so the Aborting error message is printed in the child.
When it comes to other integers, the output behaves like return 1 when it returns a positive number and return -1 when it returns a negative number. This is because, if else blocks only check if x>0 or x<0. When the function passed is a void function, then the output is the same as when f returns 1. By default, the function seems to be returning 1.

E:
Waitpid is implemented by accepting two arguments, one address and one integer. The integer is the pid of the process to be waited for. These arguments are accepted in sys_waitpid() using argint and argaddr functions.

These are passed to waitpid() function in proc.c which implements the waitpid functionality. This function is similar to the wait() function. The only difference is that the process table is traversed till the process of the given pid is found. Now, the parent relation with the current process is checked and if the passed pid is 1, then the process is init process, so return -1 is done to avoid kernel trap. Now, the rest of the code is same apart from finding the specific process.

F:

For the implementation of ps() we have added 3 more fields in_proc structure namely ctime,stime,etime denoting creation time, start time, end time(if already reached zombie state, else denoting current time). First we traverse process table and checked if the current process table entry does not assigned i.e, UNUSED. And if it is assigned then print all the required fields for that process. For finding current time we have used tick variable as implemented in sys_uptime.

G:

pinfo is implemented by taking two arguments from user program namely id(or -1) and procstat * structure. We have made 2 separate cases to handle each case separately. In case of id=-1 we simply store all required fields for current process and for other cases we simply traverse process table and once the passed id is reached store its entries in procstat structure.
Finally passed the procstat structure using copyout as implemented in wait system call.