RUSHANG V KARIA
ASU # 1206337661

CSE 598: EMBEDDED SYSTEMS PROGRAMMING
TASK 1

BACKGROUND READING ON WORKQUEUES

After going through the source code for both psmouse-base.c and serio.c and also observing where work-queues are used in the linux kernel (I clicked on queue_work in lxr-free electrons and saw the list where the function is called).

I can say that slow activities (--> input/output) is usually done using workqueues. Also interrupt handlers use workqueues to schedule the work so that the bottom half can be executed quickly and the interrupt can be serviced quick. The reason for using the workqueues is that they can run in process context unlike tasklets. Thus these functions can sleep and are pre-emptible. Moreover these works can be scheduled on as many processors as there are. This is why the kernel is slowly going from tasklets to workqueues (this is why workqueues were created).

NOW REGARDING PSMOUSE-BASE

The reason why a workqueue is created for psmouse is that this queue_work function is called in an interrupt handler. The code lies in the bottom half and therefore needs to exit quickly. Since a mouse may invalidate a protocol if idle for long periods of time or may lose synchronization occasionally, it needs to be resynced. On observing the code for psmouse_resync() i found that is is very large, and if it were to be included inline, the bottom half of the handler would become very slow.

A single threaded workqueue is used so that a single thread is created for the system (all the processors). Else the number of threads increases.
psmouse_queue_work() is the function that queue's the work although I do not understand why the kernel develepors chose to create a seperate function!!

Therefore queue_work is used so that the less imp and delay tolerating work can be done later

SERIO.C

The reasoning pretty much follows what I mentioned for psmouse but there is one very big difference. Here the work is queued in an event handler (serio_queue_event()) and not in an interrupt handler. This event handler handles various "unlikely" events like Registering Ports, Rescanning ports (each of these functions are called barely twice in the code).

However, serio_queue_event() uses a spinlock . Therefore it should not hold the processor for very long. (This spinlock is used exclusively in the module). Moreover the events that are queued are mostly asynchronous.

A POINT TO NOTE
Therefore the kernel developers have decided to use queing in serio.c . One additional point to note is that queue_work (serio.c) uses the system queue rather than creating a queue (psmouse). This I think is because serio.c is handling low level events and psmouse base might be dealing with abstractions of the device. Thus I think it used the system queue for priority scheduling.