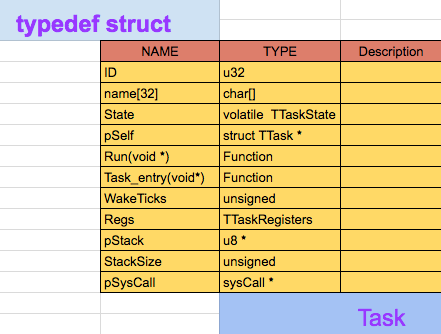
Pi-OS scheduler Design

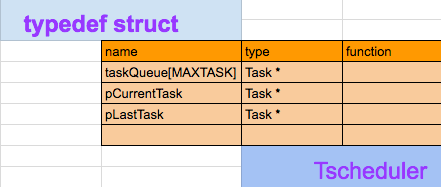
**Overview**: This passage describes the basic design of pi-OS scheduler, including important data structures, working flow, scheduling algorithms. All the tasks in the OS can be described by task structure, and can be put in the task queue. Scheduler will keep track of all the task in the queue through task structure. Scheduler is responsible for managing task state, scheduling task, executing task.

## Important data structures

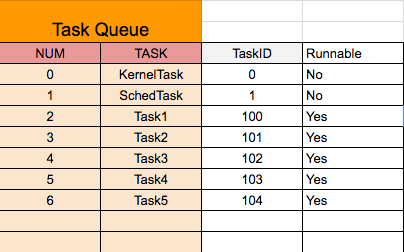
1. Task descriptor



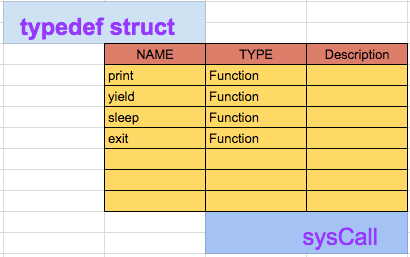
1. Scheduler structure



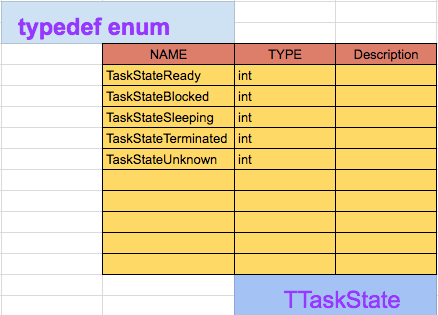
1. task queue



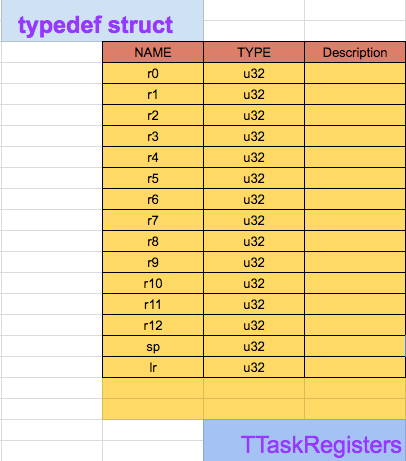
1. System call



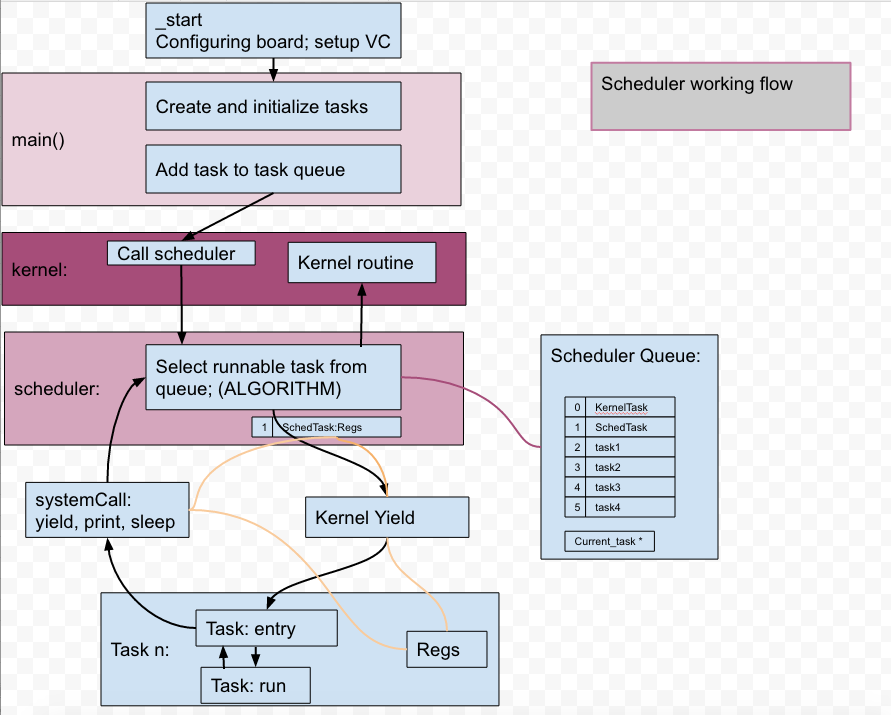
1. Task state



1. Task Register Structure



## 2. basic working flow



## 3. Scheduling algorithm

1. First In First Out (Default)

This scheduling algorithm will sect runnable task from the queue in order(except kernel task and scheduler task). When one task is over, return control to scheduler, scheduler would select the next runnable task. When there’s no runnable in the queue, scheduler stops.

2) Priority Scheduling

This scheduling algorithm will sect runnable task according to priority. User task has a priority range of 20~99. The smaller value, the higher priority. Scheduler will always select the runnable task with the highest priority in the queue to run. When one task is over, return control to scheduler, scheduler would select the next runnable task. When there’s no runnable in the queue, scheduler stops.

## 4.Key implementation

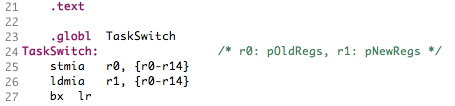
1. Task switching

Implemented using assembly code. KernelYield or Yield would call this function to switch context, saving current context in pOldRegs, and switching to new context in pNewRegs.

pNewRegs contains the target code we want to reach(in **lr**), parameter is in r0. This is easy to understand.

When someone(the CALLER) calls TaskSwitch, the CALLER’s return address will be saved in **lr**. So stmia would save **lr** (CALLER’s return address) in pOldRegsters. This is quite important if we want our old task **restart** at *the line right after calling TaskSwitch*, when it starts again by task switching.





2)

## 4. Advanced scheduling features

## 5. Component constraints

1. Static variables or structures should not be accessed directly from functions in other source file. They can be accessed by function like *Get\_variable\_name() , which return the pointer of variables.*

2) user task should never call system functions directly, they must call system functions (like print, sleep, yield) through system call. Currently system call is implemented by function pointers in Task.

## 6. Some implementation techniques and suggestion