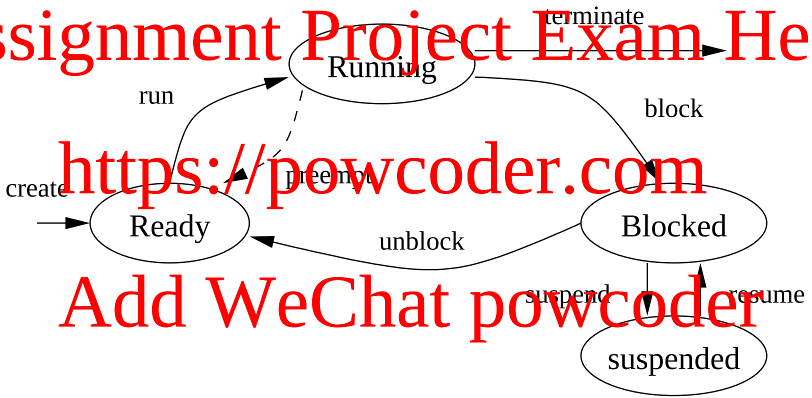
Timeline Scheduler

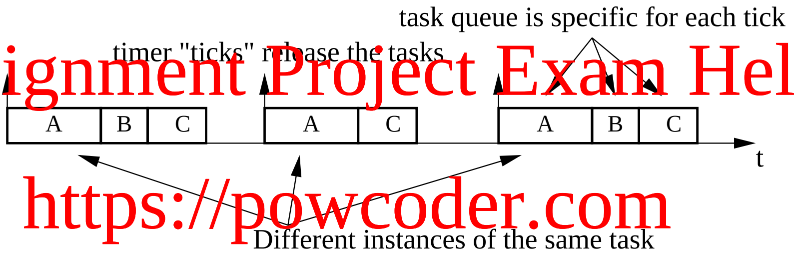
Dr. Bystrov

School of Engineering
Newcastle University

Task model (generic)

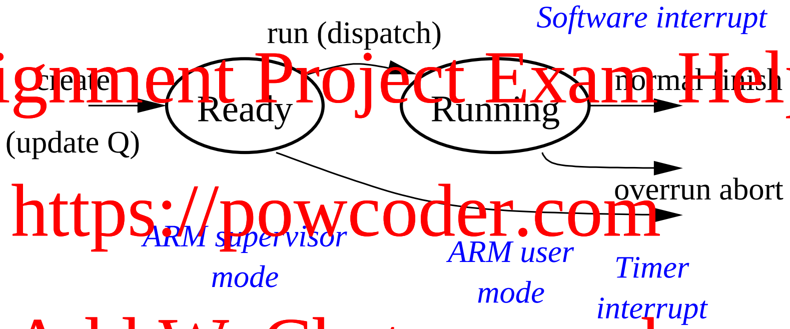


Simple Timeline TTS



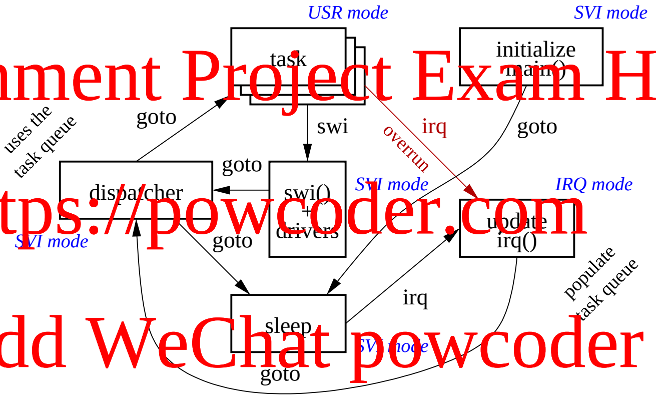
- Short periodic tasks, deadlines coincide with the next release
- Cooperative
- Static, off-line, trivially optimal
- Very simple,predictable, fast
- Overruns, only short periodic tasks

Timeline TTS task model



- Task queue is updated on each tick, which “creates” the tasks. Their code is stored in ROM.
- An overrun may happen either either when the task is running or while it still pending (in the previous task).

Timeline TTS state graph



- This is an FSM model, not a subroutine call tree!
- “Naked” functions, goto, stack init in each block.

Timeline TTS C-code

- ARM platform – supervisor (SVC), interrupt (IRQ), user (USER) modes use different registers
- "Naked" functions – no prologue/epilogue, stack needs to be reinitialised at the beginning
- No-return calls
- The only way to switch from USER to SVC is to use the software interrupt instruction
- FSM (goto) programming paradigm
- Trees of function calls only exist locally inside the blocks, e.g. inside the tasks

Short periodic task programming

- We need to implement a continuous process
- ... but only short periodic tasks are available

  - isolate the main loop
  - represent it's body as a task
  - "stitch" the task instances together with static data

Proportional function


```
void proportional()
{
  int *x=IN_PORT, *y=OUT_PORT; // IO addresses
  const int k;
  *y=k*(*x);
}
```


- adjust the data types if int is not appropriate

Integration over time


```
int a=0; // accumulator

void integrator()
{
  int *x=IN_PORT, *y=OUT_PORT; // IO addresses
  const int dt=5; // run period e.g. 5 time units
  a=a+(*x)*dt;
  *y=a;
}
```

- adjust the data types if int is not appropriate

Differentiator

```
int old_x=0;

void diff()
{
  int *x=IN_PORT, *y=OUT_PORT; // IO addresses
  const int dt=5; // run period e.g. 5 time units
  *y=( (*x) - old_x ) / dt;
  old_x=(*x);
}
```


- Danger! Differentiation is sensitive to the discretization and jitter noise.
- Instead of slow division one may use multiplication by 1/dt; the data types may need to be adjusted

Toggle


```
int tog=0;

void toggle()
{
  tog ^=1;
}
```

Sawtooth functions


```
int counter=0;

void sawtooth_rising()
{
  if (counter == MODULUS)
    counter=0;
  else
    counter++;
}
```

PWM

```
int counter=MODULUS;

void pwm()
{
  int *y=OUT_PORT; // IO addresses
// sawtooth function
  if (counter == MODULUS)
    counter=0;
  else
    counter++;
// threshold function
  if (counter < LEVEL)// LEVEL is duty_cycle*MODULUS
    *y=1;
  else
    *y=0;
}
```

Arbitrary functions of time

```
int counter=0;
int wave_data[LENGTH]=
{...}; // define the function as a table
void waveform()
{
  int *y=OUT_PORT; // IO addresses
  *y = wave_data[counter];

  if(counter==LENGTH)
    counter=0;
  else
    counter++;
}
```

PID controllers

- The mouse or the racing car project – one can use a single PIC to control several independent subsystems:
  - speed control
  - direction
  - Segway-style balancing

- Add to these several non-PID tasks
  - sensor data processing
  - labyrinth solving

Conclusions (revision)

- Generic task model
- Timeline TTS – task model
- Timeline TTS – scheduler model, overruns, task algorithms