Analogous CPU Project

Matthew C. Lindeman

New Mexico Institute of Mining and Technology

May 5, 2022

Project Overview

- Analogous Virtual CPU
- “On the fly” scheduler implementation and testing

Scheduling Algorithm Interface

- Proprietary - Good Luck
- FOSS Alternatives
  - BSD Kernel
  - Linux Kernel

Serious Look at Linux Kernel

Pros:

- End user ease of access
- Personal/Enterprise Hardware Improvements

Cons:

- **TIME**
  1. 13 files entirely re-written
  2. All files that interacted with current scheduling need updated/new data ports
  3. Older Version (2.6.3)? (Lottery Scheduler for the Linux Kernel, Mejia, Morales-Betancourt, Patki)
- Inheritance From First Process

Decided Upon Solution

Make an integrated analogous virtual CPU.

Preconditions and Postconditions

Expected input:

- A Process List
- A CPU
- Scheduling Algorithm and Related Data Structures

Project's Desired Output:

- Performance Statistics Such as:
  1. Average rate of completion with respect to all classes of processes (such as memory access and IO)
  2. Total Memory Consumption
  3. etc.

Basis Of Stochastic Scheduling Algorithm

A basis for a stochastic scheduling algorithm must have the following formally:

1. α - The hardware on which the algorithm will run.
2. β - The data structures that the processes have relating them to the scheduling algorithm (can be the empty set!)
3. γ - The implementation of the scheduling of the list of processes (the algorithm itself).

Ideal Requirements of Analgous CPU

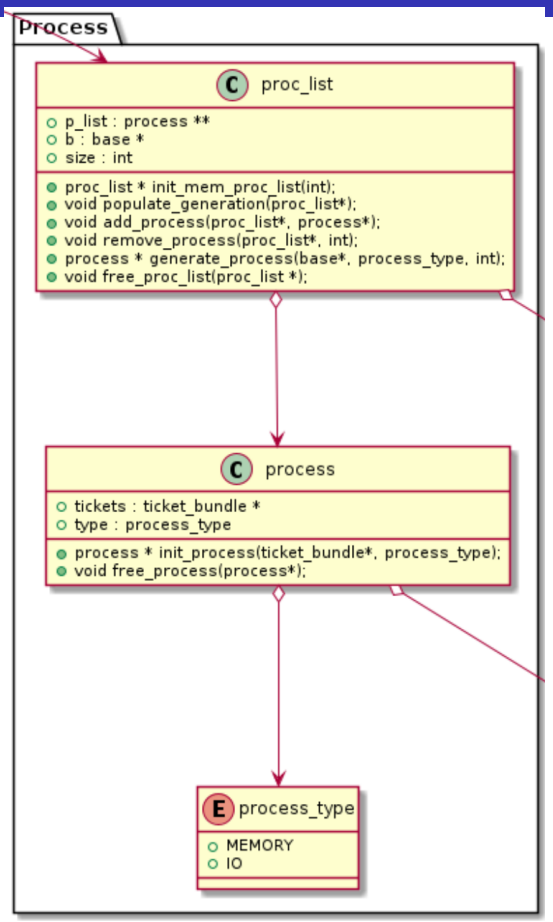
- Be opaque to the user with respect to anything other than the scheduling algorithm interface and data structure (i.e. the user can implement γ without worrying about other environmental factors).
- Have a full list of parameters for the user to test that would impact the way processes are fed into the virtual CPU.

Overview

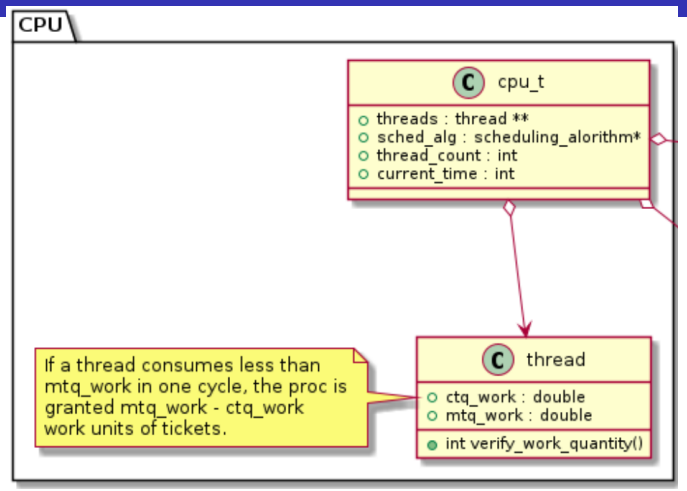
1. Compare Fair Scheduling and Lottery Scheduling Algorithms
2. Variable Parameters
3. Analogous $\Longrightarrow$ reduction of dimension in some manner. Here we just treat work/memory partitions as scalars not vectors.

Process

1. Encourages Genericity With Respect to CPU/Scheduling Algorithm
2. Process List only requires Scheduling Data Type and a max size



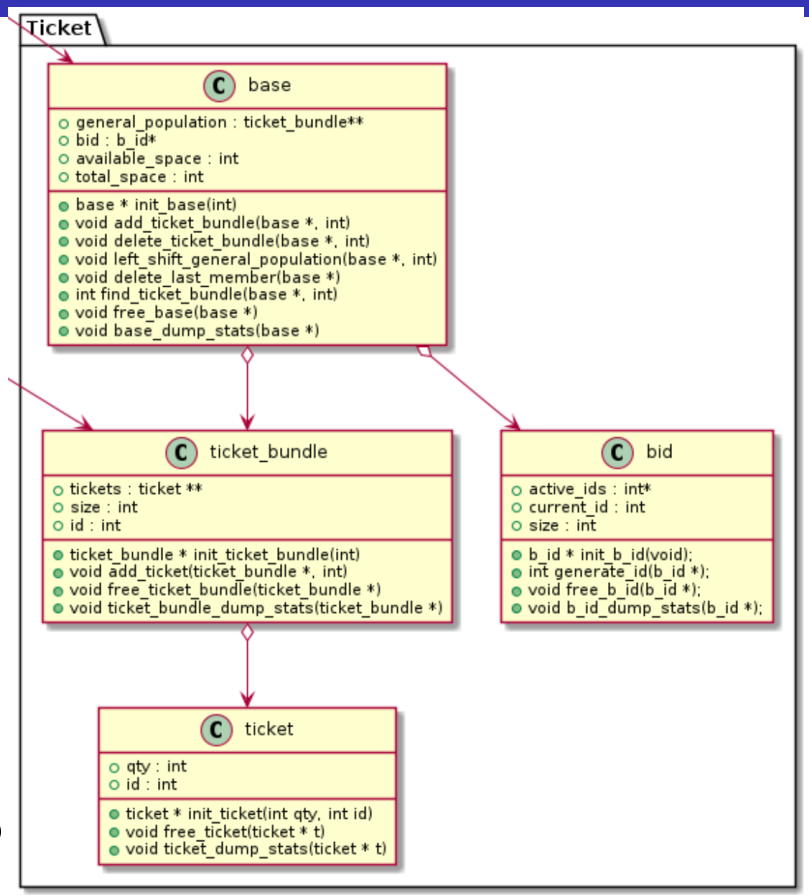
CPU



1. Interfaces only with a generic scheduling algorithm and process list.
2. Worth Noting Here Thread completion quantity per time per time quantum is a **scalar**

Scheduling Specific

1. An example structure of how a user would implement an algorithm to work with the CPU simulator.
2. ∃ universal collective data structure to interface with proc_list
3. ∃ sub-structure for each process to point to (make SA opaque to the user!)



Details about Status of Project

This tool is still in development. The CPU, Process List, Lottery Scheduling data structures and most logic for the lottery scheduler are implemented.

- Currently a primitive working ecosystem.
- Data - the lottery scheduler itself still fails some unit tests thus the data produced cannot be trusted 100% and is not included here.
- Hosted: https://github.com/millipedes/Scheduling-Simulator/
- Lang: C
- Scope $\approx$ 1000 lines of code across 25 files.

Potential Future Research Potential

1. Extensibility With Respect to Scheduling Algorithms.
2. Additional differentiations to make between process. Such as user space/kernel space explicit differentiation.
3. Implementation of opaque data statistics recording engine.