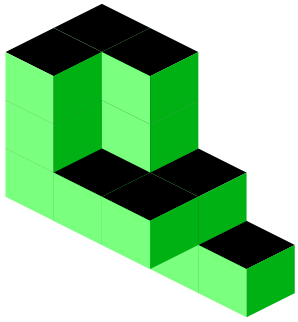
شبیه سازی

کامپیوتری



گزارش

پروژه

محمد ابول نژادیان

۹۸۱۰۳۸۶۷

محمدامین کرمی

۹۸۱۰۵۹۹۸

دکتر

بردیا صفایی

زمستان ۱۴۰۱

مقدمه

در این پروژه، یک شبیه ساز زمان بند با مشخصات توضیح داده شده در صورت پروژه پیاده سازی شده است. ابتدا به توضیح ساختار کد میپردازیم و سپس خروجی های مورد نیاز پروژه را با یک مثال میدهیم.

ساختار کد

تمام منطق شبیه سازی در CpuSimulation پیاده سازی شده است. در این کلاس، توابعی که منطق کلی شبیه سازی از جمله جلو بردن زمان، فراخوانی JobLoader، JobCreator و Dispatcher و خروجی دادن است را انجام میدهد. چندین مورد از توابع مهم این کلاس را در ادامه توضیح میدهیم.

- **:_generate_processes**

مهم ترین تابع initialize_، همین تابع است. در این تابع بر اساس پارامتر های تعیین شده، پردازه ها ساخته میشوند و در لیست self.generated_processes قرار میگیرند. این تابع در اصل همان JobCreator است.

- **:_add_arrived_process_to_queue**

این تابع که در ابتدای هر سیکل اجرا میشود، تمامی پردازه هایی که arrival_time ها آن ها سر رسیده است را به priority queue اضافه میکند.

- **:_should_insert_processes_to_second_level**

این تابع طی هر n ثانیه که به عنوان INSERT_TO_SECOND_LEVEL_INTERVAL تعریف شده است، چک میکند که آیا k تسک در صف های طبقه دوم هستند یا نه و اگر نبودند k تا اضافه میکند. این تابع در اصل همان JobLoader است.

- **:_dispatch**

در این تابع، هر دو منطق که یکی از آنها بر اساس اولویت است و دیگری بر اساس استفاده از randomness است پیاده سازی شده است(مورد امتیازی). برای استفاده از randomness کافه است تا مقدار USE_RANDOMNESS را true کنیم. منطق اصلی آن هم این است که ابتدا صفی که باید پردازه بعدی از آن انتخاب شود را انتخاب میکنیم و سپس بر اساس سیاست آن صف که در policy آن صف تعیین شده است، پردازه را انتخاب میکنیم.

- **:_check_timeouts**

این تابع در جهت پیاده سازی مورد امتیازی زده شده است. در این تابع که در انتهای هر سیکل صدا زده میشود، چک میشود که آیا timout هیچ کدام از پردازه ها گذشته است یا نه و اگر هر کدام از پردازه ها timeout اش گذشته باشد، آن را از سیستم بیرون می اندازد.

همچنین در کنار CpuSimulation، سه کلاس مهم دیگر هم هستند:

- **:CPU**

در این کلاس، منطق کلی CPU از جمله اینکه چگونه یک پردازه را serve کند پیاده شده است. به این گونه که پردازه در حال حاضر در حال اجرا شدن و مقدار زمانی که باید اجرا شود و اینکه بعد از اتمام زمان اجرا باید به کدام صف برود (next_queue)، همه در این کلاس نگهداری میشوند.

- **QUEUE:**

این یک کلاس abstract است که سه صف FIFO و PrioriyQueue و RoundRoubin آن را extend میکنند و از قابلیت های آن استفاده میکنند. به این صورت که انتخاب پردازه بعدی از هر کدام از آنها مختلف است و RoundRobin یک پراپرتی اضافه به نام quantum_time_ دارد.
همچنین policy برای انتخاب پردازه بعدی از PriorityQueue را با توجه به توضیحات صورت پروژه، با توجه به عدد زیر به دست می آوریم.

-(process._priority.value[0] + 0.1 * waiting_time)

منطق این فرمول بر این اساس از که اولویت هر پردازه که بالاتر بود و همچنین ۰.۱ هم مقدار منتظر ماندن آن در صف را در تصمیم گیری را موثر میداند که جلوی starvation گرفته شود.

- **Process:**

منطق پردازه ها و تمام فیلد های مربوط به پردازه ها از جمله زمان تمام شدن، زمان در صف ماندن و باقی فیلد ها در این کلاس قرار خواهند گرفت.

ورودی نمونه و روند برنامه

در این قسمت روند نمونه برنامه را اجرا میکنیم.

فرض میکنیم ورودی ما به شکل زیر است. [نکته: برای یکسان شدن نتایج، یک seed به مقدار ۱۰۰ تعیین کرده ایم که مقادیر شانسی، یکی شوند.)

Simulation Time	100
Process Counts	25
Interarrival Rate(X)	0.3
Service Time Mean(Y)	10
Timeout Mean(Z)	20

ابتدا برنامه میخواهد تا تعیین کنیم از طریق فایل میخواهیم ورودی ها را بدهیم یا CLI. با CLI ورودی ها را میدهیم.

```
→  simulation-project git:(master)
How do you want to read inputs?
1. CLI
2. File
1
Simulation time: 100
Process counts: 25
Interarrival rate(X): 0.3
Service time mean(Y): 10
Timeout mean(Z): 20
```

بعد از آن، simulation table مربوطه در CLI نمایش داده میشود و تمامی خروجی ها چاپ میشوند. برای مثال simulation table نمونه با غیرفعال بودن انتخاب random صف ها در لایه دوم، q1 = 1 و q2 = 5 برای صف های RR اول و دوم، و همچنین بازه های چک ۱۰ تایی برای انتقال از لایه اول به دوم و بچ های ۳ تایی (k=3) برای بردن به لایه دوم، داریم:

	Priority	Interarrival Time	Arrival Time	Service Time	Waiting Time	Served Time	Finished Time	Remaining Service Time	Finished Successfully	Timeout	Timouted?
0	Priority.NORMAL	0	0	1	0	1	1.0	0	True	12	False
1	Priority.NORMAL	4	4	12	24	12	40.0	0	True	26	False
2	Priority.HIGH	1	5	16	16	6	27.0	10	False	15	True
3	Priority.LOW	0	5	6	18	6	29.0	0	True	54	False
4	Priority.NORMAL	9	14	4	8	0	22.0	4	False	7	True
5	Priority.HIGH	4	18	2	4	0	22.0	2	False	3	True
6	Priority.LOW	0	18	4	10	4	32.0	0	True	19	False
7	Priority.LOW	11	29	2	3	2	34.0	0	True	62	False
8	Priority.LOW	2	31	23	35	6	72.0	17	False	34	True
9	Priority.NORMAL	0	31	10	3	0	34.0	10	False	2	True
10	Priority.LOW	0	31	5	16	5	52.0	0	True	29	False
11	Priority.HIGH	2	33	2	21	2	56.0	0	True	36	False
12	Priority.LOW	8	41	2	12	1	54.0	1	False	11	True
13	Priority.LOW	3	44	23	3	0	47.0	23	False	2	True
14	Priority.NORMAL	2	46	34	19	6	71.0	28	False	18	True
15	Priority.LOW	4	50	14	15	6	71.0	8	False	14	True
16	Priority.LOW	1	51	8	16	1	68.0	7	False	15	True
17	Priority.LOW	1	52	19	10	1	63.0	18	False	9	True
18	Priority.LOW	2	54	14	14	14	82.0	0	True	33	False
19	Priority.LOW	7	61	10	6	0	67.0	10	False	5	True
20	Priority.NORMAL	6	67	24	18	15	NaN	9	False	25	False
21	Priority.HIGH	4	71	7	14	1	86.0	6	False	13	True
22	Priority.LOW	2	73	1	11	1	85.0	0	True	22	False
23	Priority.LOW	6	79	1	11	1	91.0	0	True	28	False
24	Priority.LOW	4	83	1	2	0	85.0	1	False	1	True

برای این ورودی ها و پارامتر های گفته شده، خروجی های خواسته شده به شکل زیر خواهد بود:

1 . میانگین طول صف ها در طول کل زمان شبیه سازی:

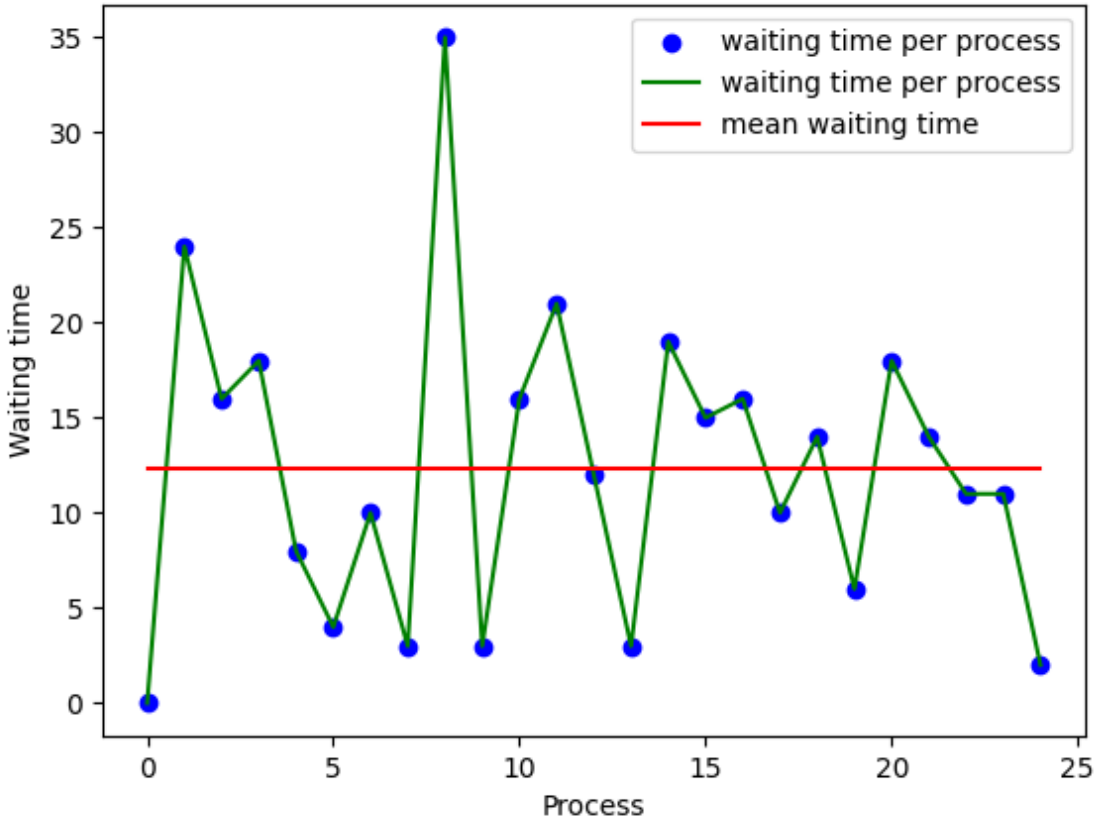
Priority Queue	1.46
Round Robin T1	0.41
Round Robin T2	0.91
FCFS	0.68

برای محاسبه این اعداد، در هر سیکل طول هر صف را ثبت میکنیم.

2 . میانگین زمان صرف شده در صف ها(به طور کلی و به ازای هر پردازه):

به ازای کل پردازه ها: ۱۲.۳۶ واحد زمانی

برای هر کدام از پردازه ها:



3 . بهره وری CPU:

۹۱ درصد

5 . درصد پردازه های منقضی شده:

۵۶ درصد

4 . پیشنهاد برای بهبود میانگین زمان صف ها:

میانگین زمان صف ها در حالتی که زمان کوانتوم صف های اول و دوم robin round ۱ و ۵ بود را دیدیم.
حال زمان های مختلف را بررسی میکنم:

q1	q2	Priority Queue	Round Robin 1	Round Robin 2	FCFS	Total Average
1	5	1.46	0.41	0.91	0.68	**0.865**
5	10	1.62	0.95	0.9	0.02	**0.872**
10	20	1.91	1.58	0.54	0	**1.007**
1	3	1.47	0.52	0.62	1.01	**0.905**
2	5	1.87	0.5	0.75	0.47	**0.897**
2	10	1.47	0.73	0.89	0.45	**0.885**

نتیجه میشود برای بهبود زمان انتظار در صف های اول (RR وPriority) باید کوانتوم های کوچک در نظر گرفت و برای کم شدن زمان انتظار در صف آخر (FCFS)خوب است که این زمان را بیشتر کنیم.

به طور کلی هم منطق درست این است که کوانتوم اول زمانی کم برای انجام کار های سریع را داشته باشد و کوانتوم دوم زمانی متوسط و نزدیک به میانه باشد. میانه زمان های سرویس دهی در این مثال ۱۰ بود و ممکن است کار های کوچک با زمان ۱ داشته باشیم که این مقدار بهتری است.
پس کار هایی که بیشتر طول میکشند سنگینی میکنند و به صف های بعدی لایه دوم میروند.