Especialização do Mecanismo

Sistemas Operativos

1 O que é preciso fazer?

1.1 Consistência de Dados

Assim que o servidor reinicia tenta restabelecer a posição anteriormente estabelecida. Isto pode ser feito pelo ficheiro `log.idx`, bastando, para tal, assumir que cada entrada do ficheiro `log.idx` tem um tamanho fixo.

Esse tamanho pode ser simplesmente `3 * sizeof(unsigned long)`, pois só precisamos, para cada entrada de 3 números:

1. *ID* da tarefa.
2. Offset da tarefa indicada no ficheiro de output de logs.
3. Tamanho em bytes do output.

Desta forma, assim que o servidor inicia, o primeiro comportamento que deve fazer é abrir o ficheiro `log.idx` com as permissões `O_CREAT | O_RDWR`, de denotar que este ficheiro vai ser binário, incompreensivel do ponto de vista humano.

Para redefinir o ID da tarefa, basta, apos abertura do ficheiro acima, fazemos `x = lseek(fd, 0, SEEK_END)` isto vai posicionar o *fd* no *end of file*, ou seja, agora escrita funcionam como append.

Em `x` fica o valor do offset do end of file, quando lá chegamos, por essa razão. Para descobrir o próximo ID basta fazer `x / N`, sendo `N` o tamanho de cada uma das entradas. Assim sendo, se encontramos `x = 12`, sabendo que cada entrada ocupa `N = 6`, então veremos que o próximo id deverã ser `x / N = 12 / 6 = 2`, logo, é imediato o novo id. Garante-se assim a consitencia dos dados.

1.2 Como ler um resultado do ficheiro de logs?

Para ler um dos outputs, primeiro precisamos de saber o seu offset e o seu tamanho no ficheiro de logs, esses valores eencontram-se no ficheiro `logs`, porém, ainda não temos nenhuma forma de eficientemente aceder estes. Isto porque, garantir que o ficheiro `log.idx` é continuo em memória é uma falacia, porque

um dos pedidos pode terminar primeiro, e nesse caso seria colocado no ficheiro `log.idx` antes de um ficheiro que lhe possa ter precedido.

Uma opção, seria procurar ao longo do ficheiro, porém isto é pessimo em termos de desempenho. A alteranativa mais viável seria tentar escrever em memoria todos os indices na posição necessária, mesmo que não tenha terminado tarefas que a antecedem.

Isto poderá ser feito colocando o ficheiro imediatamente na posição necessária. Porém, pode acontecer o caso de o offset que pretendemos colocar ser superior ao offset maximo do ficheiro que temos em aberto, neste caso pode ser necessario imbutir um padding até chegarmos ao offset necessario, talvez com nulos.

É também útil possuir uma cache de valores de recentemente utilizados, com um teto máximo de `CACHED_UNITS`. Isto porque será muito mais eficiente aceder diretamente à cache do que estarmos sempre a trabalhar no ficheiro.

É preciso definir qual será o formato deste ficheiro, podera ser util fornecer um ficheiro de codigo só para este efeito, com o intuito de trabalhar no ficheiro de logs, vê-se isto mais a frente.

Por isso, asumindo que já temos uma forma de armazenar logs de output, resta pensar no historico.

1.3 Como implementar o histórico?

Relativamente fácil, é só um ficheiro no qual vamos dando append à medida que os processos vão terminado. Para darmos append precisamos de saber de que forma é que o processo terminou, sendo que existe 3 formas no qual ele pode terminar:

```
#define COMMAND_SUCESS        0
#define COMMAND_PIPE_TIMEOUT  1
#define COMMAND_EXEC_TIMEOUT  2
```

Ademais, requer também o conhecimento do comando que foi executado, sendo que o output no ficheiro de histórico deverá então ser:

```
#2, concluida: ./a.out | ./b.out
#3, max inactividade: prog1 | prog2 | prog3
#4, max execução: prog2 | prog3
```

O mapeamento entre os códigos de erro e a respectiva string deverá ser direto.

1.4 Ponto de situação

Até agora temos o seguinte compreendido:

- Como fazer ficheiro de logs com indexação.

- Como fazer histórico.

Falta portanto perceber como vai funcionar a aplicação em si, que de facto é o menor dos nossos problemas.