Hw4: Memory Manager Simulation

Results

TLB Policy = `LRU`

EAT	`FIFO` + `GLOBAL`	`FIFO` + `LOCAL`	`CLOCK` + `GLOBAL`	`CLOCK` + `LOCAL`
Process A	164.758	164.980	164.758	164.980
Process B	163.709	163.144	163.709	163.522

Page Fault Rate	`FIFO` + `GLOBAL`	`FIFO` + `LOCAL`	`CLOCK` + `GLOBAL`	`CLOCK` + `LOCAL`
Process A	0.723	0.774	0.723	0.774
Process B	0.665	0.700	0.665	0.694

TLB Policy = `RANDOM`

EAT	`FIFO` + `GLOBAL`	`FIFO` + `LOCAL`	`CLOCK` + `GLOBAL`	`CLOCK` + `LOCAL`
Process A	164.534	164.980	164.534	164.980
Process B	162.953	162.953	162.761	162.953

Page Fault Rate	`FIFO` + `GLOBAL`	`FIFO` + `LOCAL`	`CLOCK` + `GLOBAL`	`CLOCK` + `LOCAL`
Process A	0.723	0.774	0.723	0.774
Process B	0.665	0.700	0.665	0.694

Analysis

TLB Replacement Policy: `LRU` | `RANDOM`

在 TLB 快取已滿的情況下，只要有新的 reference，就必須移除一筆舊的 entry。

- **LRU**: least recently used，移除最久沒用的entry
- **RANDOM**: 亂數移除一筆entry

LRU 的設計理念是要留下最常使用的 entries，而亂數移除就有可能踢除常用的 entry 而造成 EAT 變慢。從實驗數據上來看，process A 在 LRU 的情況下 EAT = 164.9，但是使用 random 的話，EAT 就降到只剩 163.5，代表使用 LRU 的確能加速平均存取時間。

Page Replacement Policy: `FIFO` | `CLOCK`

在 free frame list 已滿的情況下，只要有新的 page，就必須將一個舊的 page 踢除到硬碟。

- **FIFO**: 踢除最先進入 memory 的 page
- **CLOCK**: clock 依序循環檢查 free frame list 中 page 被存取的狀態。假如 page 在下一次被 clock 檢查到前有被 reference，那 clock 就會跳過它一次。 clock 會踢除見到的第一個 referenced 狀態為 false 的 page。

雖然理論上因為 CLOCK 有對使用頻率做篩選，其存取時間表現會比 FIFO 好，但是對於這次的資料集 CLOCK 的 EAT 和 page fault rate 都和 FIFO 相同。如果換一個測資應該可以避免這個狀況。

Frame Allocation Policy: `GLOBAL` | `LOCAL`

- **GLOBAL**: 在選擇踢掉哪個 page 時，所有 page list 中的 process 的 page 都有可能被選到
- **LOCAL**: 只能踢除目前 reference 到的 process 的 page

因為 global 可以踢除任何 process 的 page，所以理論上能平衡不同 process page 的數量，不會造成 local 其中一個 page 卡住太多 pages 的狀況，因此 global 有較好的 page fault rate。從實驗數據上來看，global 的 平均 page fault rate 低於 local 約 5%，EAT 則相同。