Trabalho S.O. - Memória Virtual

Daniel Gunna, Felipe Coelho

June 7, 2016

1 Introdução

Para substituir a política utilizada pelo Nanvix para a substituição de páginas (First-in-first-out), utilizamos o algoritmo NFU (Not frequently used) com aging. Nas demais seções, descreveremos as mudanças realizadas no código do Nanvix para a implementação dessa política.

Escolhemos o algoritmo NFU com aging por ser o algoritmo que mais se aproxima do algoritmo LRU (Least Recently Used) porém, com uma solução menos custosa.

2 Update Page Table

No arquivo mm.h foi definido o protótipo de uma função: "void UpdatePageTable". Essa função será responsável por atualizar a tabela de página e as frequências a cada determinado intervalo de ciclos. Essa função foi implementada no arquivo paging.c, e pode ser visto a seguir.

```
void updatePageTable( ){
        struct pte * pag; //Page table reference
        //Loop para percorrer frames
        for (int i  = 0 ; i < NR_FRAMES ; i++){
                //Pega o endereco da linha da pagina do
                    processo corrente na Page Table
                pag = getpte(curr_proc,frames[i].addr);
                //Fazer o processo de aging desloca o
                    contador de frequencia para direita
                frames[i].frequence  = frames[i].
                    frequence >> 1;
                int aux = pag->accessed; //Pega o bit R
                    da pagina
                //Empurra para o bit mais significativo
                    para fazer incluir no contador
                aux =  aux << 31;
                //Faz um or  para adicionar o bit R ao
                    contador
                frames[i].frequence |= aux;
        }
}
```

3 Alterações no Clock.c

No arquivo clock.c, na função do_clock( ) foi adicionado um trecho de código que a cada 100 ticks (2 Quantums) a tabela de páginas vai ser atualizadas, ou seja, os bit R's serão somados a frequência das páginas.

A implementação pode ser observada a seguir.

```
PRIVATE void do_clock(){
        ticks++;
        //Variavel global criada para contar os ticks
        counter++;
        //Se o contaodr chegar a 100 quer dizer que
            houveram 100 ticks ou 2 quantuns
        if(counter == 100){
                //zerar contador para nova contagem
                counter = 0;
                //Atualizar tabela de paginas chamando a
                    funcao definida em mm.h e implementada
                     em paging.c
                updatePageTable();
        }
        if (KERNEL_RUNNING(curr_proc)){
                curr_proc->ktime++;
                return;
        }
        curr_proc->utime++;
        /* Give up processor time. */
        if (--curr_proc->counter == 0)
                yield();
}
```

4 Alterações no Paging.c

No arquivo paging.c, na struct do frame foi adicionado um campo chamado frequence para contar a frequência do uso da página. A struct modificada pode ser vista a seguir.

```
PRIVATE struct
{
        unsigned count;      /**< Reference count.     */
        unsigned age;        /**< Age.                 */
        unsigned frequence; /**< Frequencia com que a
            pagina e usada.*/
        pid_t owner;         /**< Page owner.          */
        addr_t addr;         /**< Address of the page. */
} frames[NR_FRAMES] = {{0, 0, 0, 0, 0}, };
```

Além disso, no arquivo paging.c, na função allocf(), a única mudança relevante que foi realizada foi que, na busca por um frame a ser substituído, o frame com a menor frequência é selecionado. O código pode ser visto a seguir:

```
PRIVATE int allocf(void){
        int i;       /* Loop index.  */
        unsigned freq; /* Menor frequencia. */
        #define OLDEST(x, y) (frames[x].age < frames[y].
            age)

        //Variavel para guarda o valor i do frame de
            menor frequencia
        freq = 0xFFFFFFFF;
        /* Search for a free frame. */
        for (i = 0; i < NR_FRAMES; i++){

                /* Found it. */
                if (frames[i].count == 0)
                        goto found;

                /* Local page replacement policy. */
                if (frames[i].owner == curr_proc->pid){
                        /* Skip shared pages. */
                        if (frames[i].count > 1)
                                continue;

                        /* Proucurar pela pagina menos
                            frequente. */
                        if (freq == 0xFFFFFFFF  || frames
                            [i].frequence < freq)
                                freq = i;
                }
        }
        /* No frame left. */
        if (freq  ==  0xFFFFFFFF)
                return (-1);

        /* Trocar frame menos frequente . */
        if (swap_out(curr_proc, frames[i = freq].addr))
                return (-1);

found:
        frames[i].age = ticks;
        frames[i].count = 1;
        return (i);
}
```