Segment	base	bound	protection
Code (00)	۴۰k	۱۲k	r-x
heap (01)	۷۰k	۲k	rw-
Stack (10)	۵۸k	۲k	rw-

(A) ①

نحوه رشد برای Stack به صورت صفر (grow positive = 0) و برای بقیه موارد ۱ است.

(B) ۱۷ بیت داریم ⇐ بیت ۱۶ و ۱۷ شماره Segment

$0x1cc \rightarrow$ ۱۷ بیت $\rightarrow$ 00000 0001 0000 0000 (Code) $\rightarrow 40kB + 0x1cc = 0xA1cc$

$0x1FA1c = 1\,1111\,1010\,0001\,1100$ (Stack) $\rightarrow 58kB - (32kB - (40kB + 16B)) = 56kB + 16B$

(C) وارد قسمت کدی شود $\rightarrow 44kB = 58 - (32 - 18)$ Stack $\rightarrow 0x18Acc$

می‌توان این Segment را به فضایی مانند 92kB منتقل کرد که فضای خالی داریم

Segment	base	bound	protection
Stack (10)	~~58k~~ 92k	~~xk~~ 12k	rw-

$0xbccc \rightarrow heap \rightarrow 20 + 12 = 32kB$

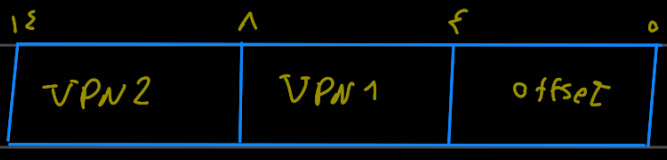
باید 12kB افزایش اندازه دهد اما چون امکان افزایش فضا و همچنین انتقال آن نیست برنامه متوقف می‌شود.

Segment	base	bound	protection
heap (01)	۷۰k	~~۲k~~ 12kB	rw-

(2)

(A

اندازه هر صفحه = ۴۲ → ۱۵ بیت → Paging ۲ سطح ⇒



0x۴a۲b → 0 1110 10 0011 11 0011

↳ ۱۴ → ۲۰ = PD در خانه ۱۴ → 0x۸۹ → 1 0000 0001 (9)۱۰

Valid

با توجه به VPN1 = ۱۷ ⇐ صفحه شماره ۹ و خانه شماره ۱۷ ⇐ 0xba ⇐ 1 0111 0100 (۱۱۸)۱۰

Valid

حال آدرس صفحه آخر = ۱۱۸ را داریم که باید به کمک offset برویم

و محتوای صفحه ۱۱۸ و خانه ۲۷ امش را بخوانیم ⇐ 0x۷b

چون بیت های سمت چپ همواره ۱ بودند پس این آدرس Valid است.

(B مانند مثال قبل عمل می کنیم ⇐ ۱۵ بیت سمت چپ ⇐ ۱۲ ⇐ ۱۲ امین خانه

صفحه شماره ۲۰ ⇐ 0x۳۴ ⇐ 0 0000 11 0100 ⇐ چون Valid نیست ⇐

Not Valid

این آدرس در فضای این پروسس نیست و فرایند ترجمه خاتمه میابد

③ به ازای segsize و Page size ما overhead داریم همچنین تعداد رکورد های

$$\frac{Segsize}{Page size} = Page\ table$$

$$overhead = (Segsize - Program size) + (\epsilon \times record)$$

که در آن Programsize حجم برنامه ما است. حال کافی است جایی که مشتق تابع برابر صفر می‌شود را پیدا کنیم تا overhead کمینه به دست آید.

$$overhead_{AV} = \frac{\int_0^{2^{32}} overhead\ dx}{2^{32}} \rightarrow \frac{d\ overhead_{AV}}{dx} = 0$$

⟸ مقدار کمینه برای اندازه هر صفحه

④ در ابتدا برای binary file و stack تا ۲ miss داریم و چون بعد از آن آدرس آنها را در TLB داریم و از روش LRU استفاده می‌کنیم دیگر برای اینها miss نداریم

امکان miss برای متغیر های global همچنان وجود دارد ⟸ تعداد کل برابر است با

$$Total = 1024^{3} + 1024^{2} + 1024 + \underbrace{2}_{stack,\ binary\ file}$$

⑤

A) اولین خانه با use = 0 را خارج می‌کند. همه use ها برابر یک اند پس یکبار همه‌ی N را می‌رود و در دور بعد PFN = 0 را که اولین را که use = 0 دارد را خارج می‌کند

$$(N+1) + 1 = N + 2$$

B) الگوریتم اول قصد دارد PFN های که متداولا استفاده می‌شوند را حذف نکند.

الگوریتم دوم هدفش سرعت بخشیدن به خالی کردن صفحات است.

افزایش ماندگاری ⟸ فاصله ۲ عقربه زیاد ⟸ کاهش ماندگاری PFN ها ⟸ فاصله ۲ عقربه کم