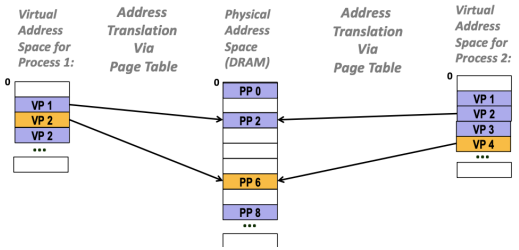
shared memory

↳ allows multiple processes to share virtual memory space

↳ very fast and good for sharing large amount of data, but no synchronization, persist without clean up



Process A

- Create shared memory segment
  - `segment id = shmget(key, size, IPC_CREAT);`
- Attach shared memory to its address space
  - `addr= (char *) shmat(id, NULL, 0);`
- write to the shared memory
  - `*addr = 1;`
- Detach shared memory
  - `shmdt(addr);`

Process B

- Use existing segment (same key, no IPC_CREAT)
  - `segment id = shmget(key, size, 0666);`
  - `addr = (char *) shmat(id, NULL, 0);`
  - `c = *addr;`
  - `shmdt(addr);`

those are different

physical address space is same

producer = produces information

```
main() {
    char c;  int shmid;   key_t key=5678;
    char *shm, *s;
   /*  Create the segment. */
    if ((shmid = shmget(key, 27, IPC_CREAT | 0666)) < 0) {
              printf("server: shmget error\n");
              exit(1);
    }
 /* Attach the segment to our data space. */
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        printf("server: shmat error\n");
        exit(1);
    }
/*  Output data*/
    s = shm;
    for (c = 'a'; c <= 'z'; c++)
        *s++ = c;
/*  Wait the client consumer to respond*/
    while (*shm != '*') sleep(1);
    shmdt(shm);
    exit(0);
}
```

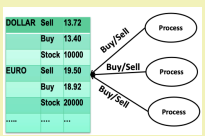
consumer= consumes information

```
main(){
    int shmid;  key_t key=5678;
    char *shm, *s;
    /* Locate the segment.  */
    if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
        printf("client: shmget error\n"); exit(1);
    }
    /*  attach the segment to our data space.*/
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        printf("client: shmat error\n");  exit(1);
    }
    /* Read what the server put in the memory, and display them*/
    for (s = shm; *s != 'z'; s++)
       putchar(*s);
    putchar('\n');
    /* Finally, change the first character of the segment to '*' */
    *shm = '*';
    exit(0);
}
```

* in shared memory, processes do not make system calls, every time, unlike pipes

currency exchange example:

```
enum currency {DOLLAR , EURO, STERLIN, POUND};
struct Currency {double sell, buy; double stock;};
int buy(struct Currency *c, double amount, double *balance) {
        if (*balance < amount*c->buy) return -1;
        *balance -= amount*c->buy;
        c->stock += amount;
        return 0;
}
int sell(struct Currency *c, double amount, double *balance) {
        if (c->stock < amount) return -1;
        *balance += amount*c->sell;
        c->stock -= amount;
        return 0;
}
```




```
#include "exchange.h"
struct Currency init[4] ={ {13.72, 13.40, 10000},
        {19.50, 18.92, 20000}, {24.551,24.552, 10000},
        {33.24, 33.25, 5000}};
struct Currency *curshared;
int main() {
        int key, i;
        // create a shared memory for 4 Currency structures
        key = shmget(EXCHKEY, sizeof(struct Currency)*4,
IPC_CREAT|0600);
        if (key < 0) { perror("shmget") ; return 1;}
        // attach it and get result in curshared pointer
        curshared = (struct Currency *) shmat( key, NULL, 0);
        for (i = 0; i < 4; i++) curshared[i] = init[i];
        shmdt((void *) curshared);
        return 0;
}
```

```
#include<exchange.h>
struct Currency *curshared;
double balance = 1000;   // initial balance
int main() {
        // get key for already created shmem
        key = shmget(EXCHKEY, sizeof(struct Currency)*4, 0);
        if (key < 0) { perror("shmget") ; return 1; }
        // attach shared memory and get address in curshared
        curshared = (struct Currency *) shmat( key, NULL, 0);
        if (curshared == NULL) return -1;
        while (fgets(line, 80, stdin)) { // trade loop
                // assume input is parsed here
                if (... "buy" ) buy(curshared+c , amount, &balance);
                if (... "sell")   sell(curshared+c , amount, &balance);
        }
        shmdt((void *) curshared);
        return 0;
}
```