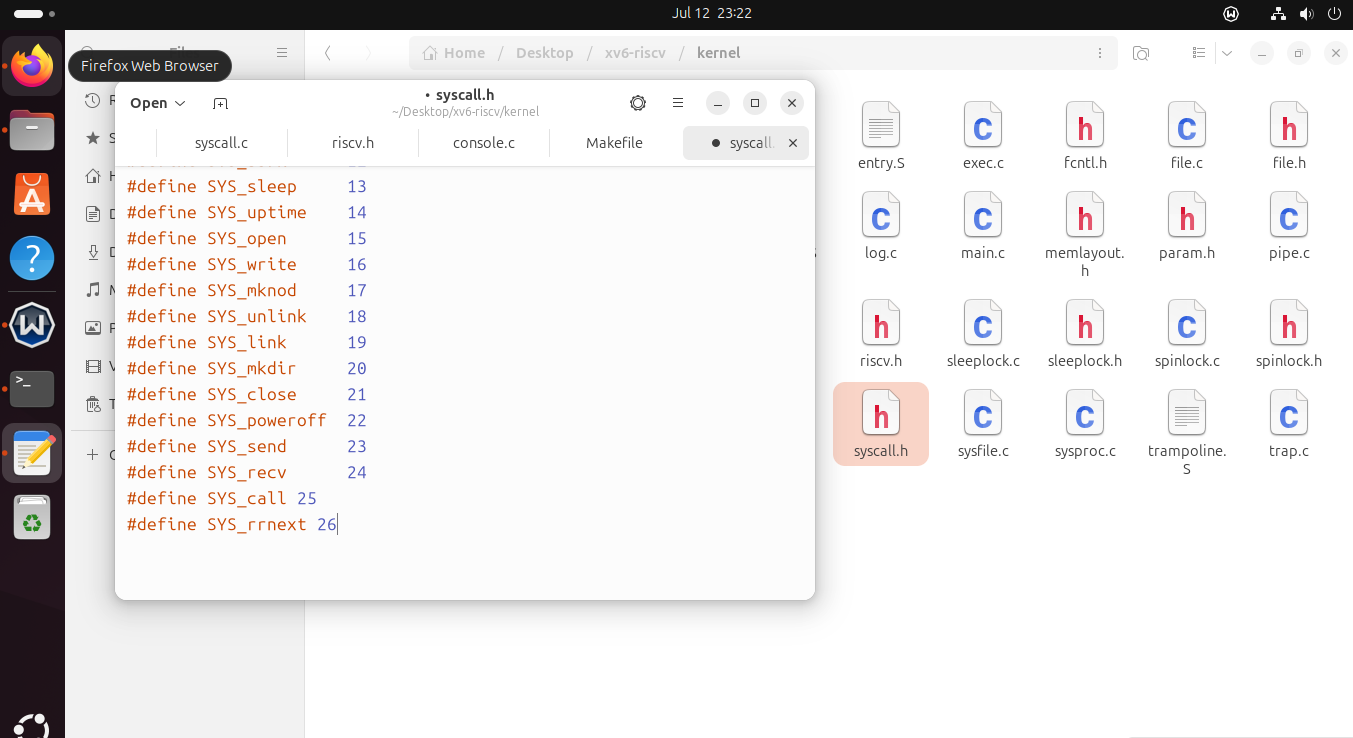
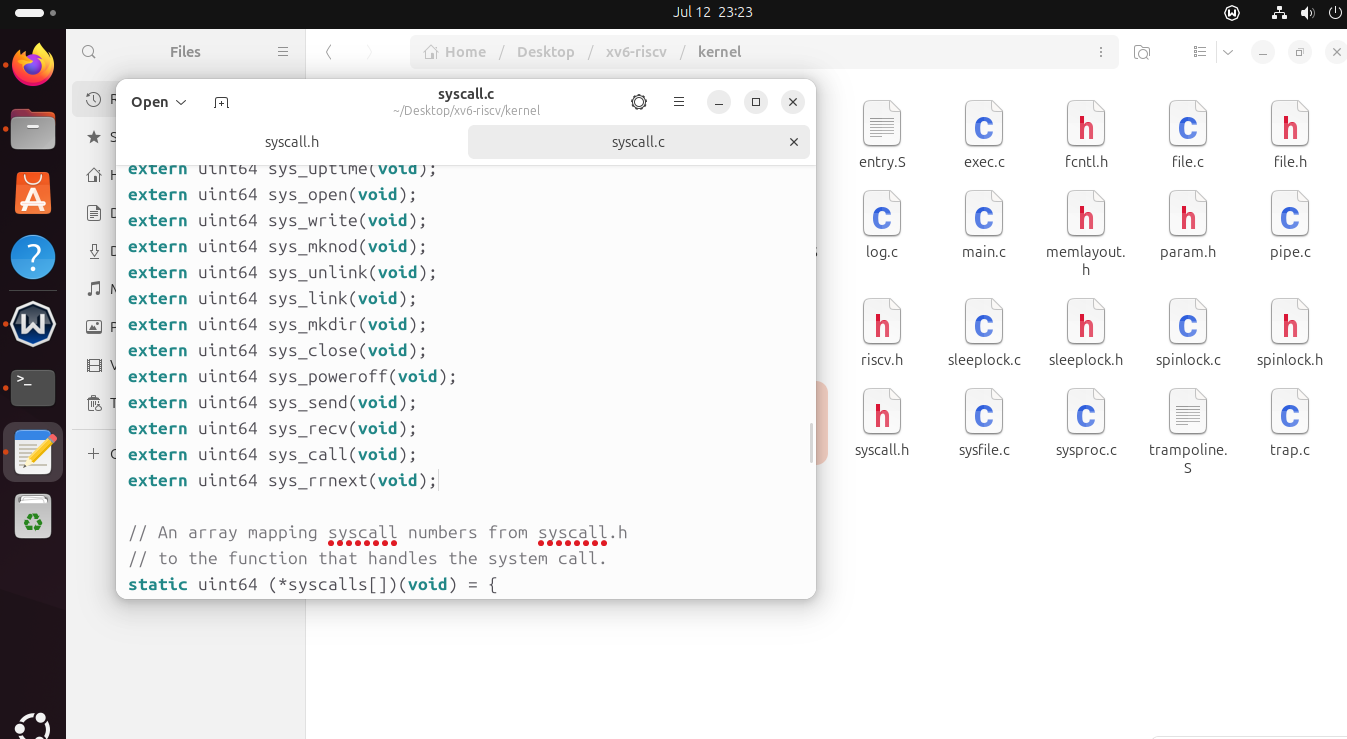
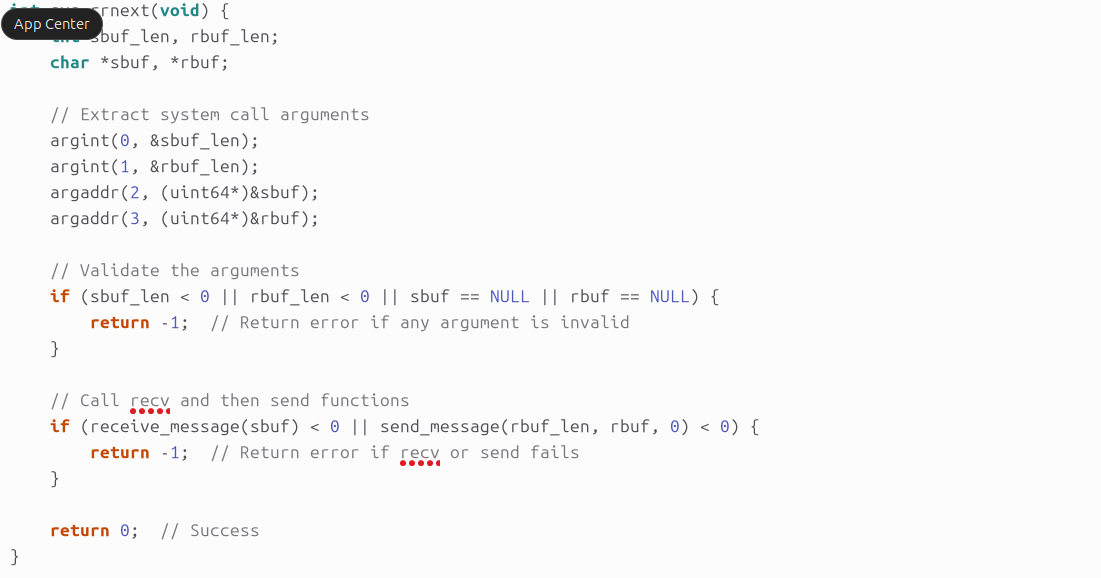
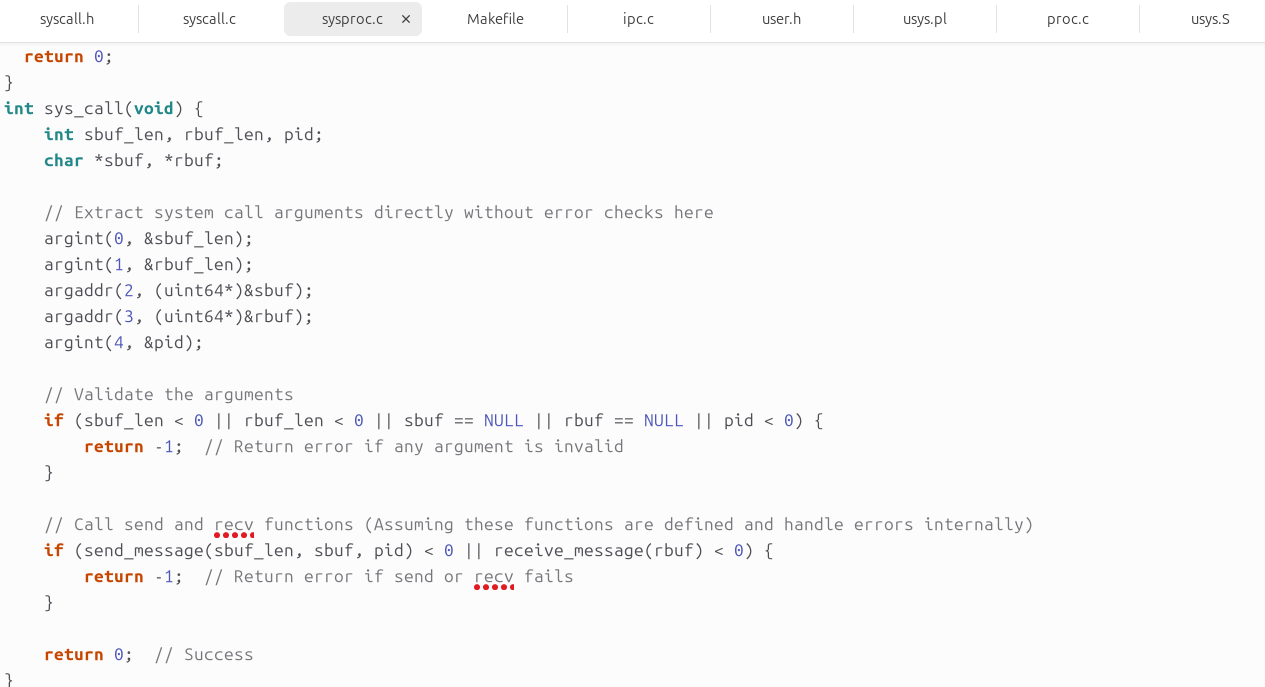
**گزارش تمرین چهار** محمدباربد امیرمزلقانی – 810102348

حالت پایه و اولیه خود را مدلی که تی ای در اختیارمان قرار داد میگذاریم که در آن تابع send و recv پیاده سازی شده بودند، در ادامه دو فراخوانی سیستمی خود را به آن اضافه میکنیم :







در اینجا مهمترین کار انجام شده را داریم که تعریف فانکشنالیتی این دو سیستم کال میباشد در ادامه به بیان جزییات هر کدام میپردازیم :

این کد یک تابع به نام sys\_call را تعریف می‌کند که برای ارسال و دریافت پیام‌ها بین فرآیندها (Processes) در سیستم‌عامل استفاده می‌شود. تابع sys\_call ابتدا آرگومان‌های ورودی سیستم‌کال را استخراج می‌کند، سپس آن‌ها را اعتبارسنجی می‌کند و در نهایت پیام‌ها را ارسال و دریافت می‌کند. اگر هر یک از مراحل با خطا مواجه شود، کد خطا بازگردانده می‌شود.

**توضیحات خط به خط کد:**

1. int sys\_call(void) {:
   * تعریف تابع sys\_call که هیچ آرگومان ورودی نمی‌پذیرد.
2. int sbuf\_len, rbuf\_len, pid;:
   * تعریف متغیرهای محلی برای طول بافرهای ارسال (sbuf\_len) و دریافت (rbuf\_len) و شناسه فرآیند (pid).
3. char \*sbuf, \*rbuf;:
   * تعریف اشاره‌گرهای محلی برای بافرهای ارسال (sbuf) و دریافت (rbuf).
4. argint(0, &sbuf\_len);:
   * استخراج آرگومان اول (طول بافر ارسال) از سیستم‌کال و ذخیره آن در sbuf\_len.
5. argint(1, &rbuf\_len);:
   * استخراج آرگومان دوم (طول بافر دریافت) و ذخیره آن در rbuf\_len.
6. argaddr(2, (uint64\*)&sbuf);:
   * استخراج آرگومان سوم (آدرس بافر ارسال) و ذخیره آن در sbuf.
7. argaddr(3, (uint64\*)&rbuf);:
   * استخراج آرگومان چهارم (آدرس بافر دریافت) و ذخیره آن در rbuf.
8. argint(4, &pid);:
   * استخراج آرگومان پنجم (شناسه فرآیند) و ذخیره آن در pid.
9. if (sbuf\_len < 0 || rbuf\_len < 0 || sbuf == NULL || rbuf == NULL || pid < 0) {:
   * بررسی اعتبار آرگومان‌ها و اگر هر کدام نامعتبر بود، -1 بازمی‌گرداند.
10. if (send\_message(sbuf\_len, sbuf, pid) < 0 || receive\_message(rbuf) < 0) {:
    * فراخوانی توابع send\_message و receive\_message برای ارسال و دریافت پیام‌ها و اگر هر کدام با خطا مواجه شدند، -1 بازمی‌گرداند.
11. return 0;:
    * بازگرداندن 0 در صورت موفقیت‌آمیز بودن تمامی مراحل.

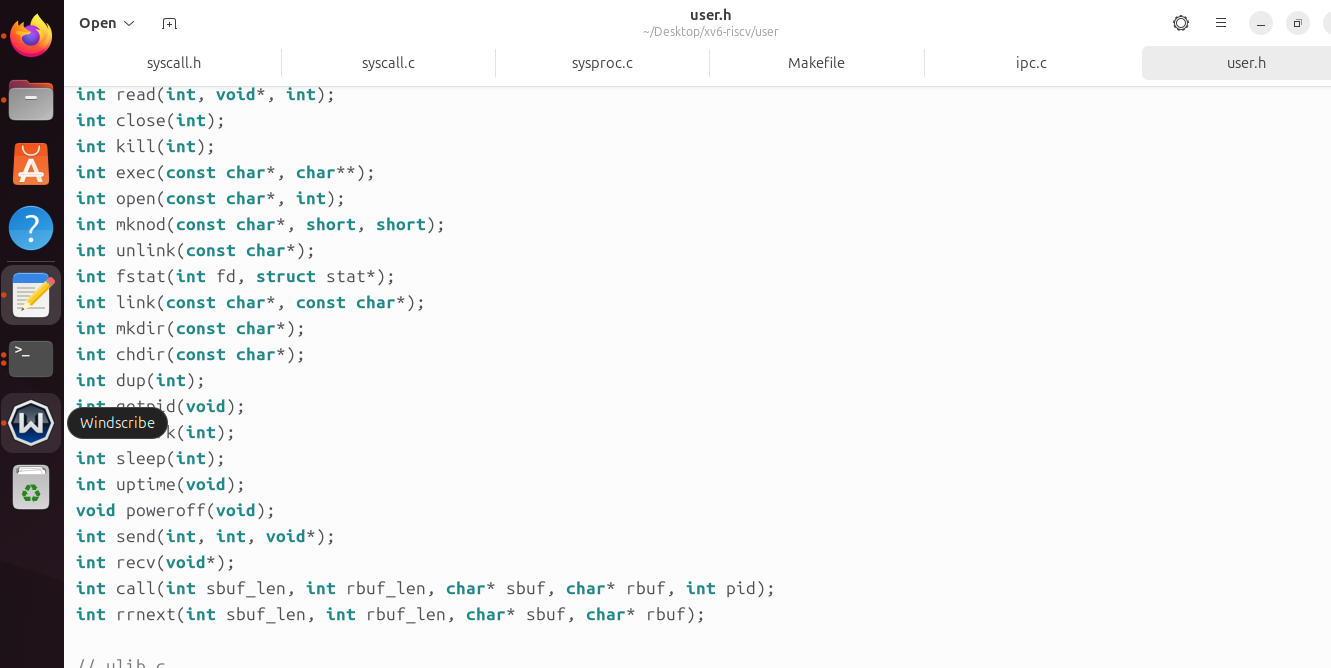
کد rrnext نیز مشابه کد قبلی یک تابع سیستم‌کال به نام sys\_rrnext را تعریف می‌کند که برای دریافت و سپس ارسال پیام‌ها بین فرآیندها در سیستم‌عامل استفاده می‌شود.

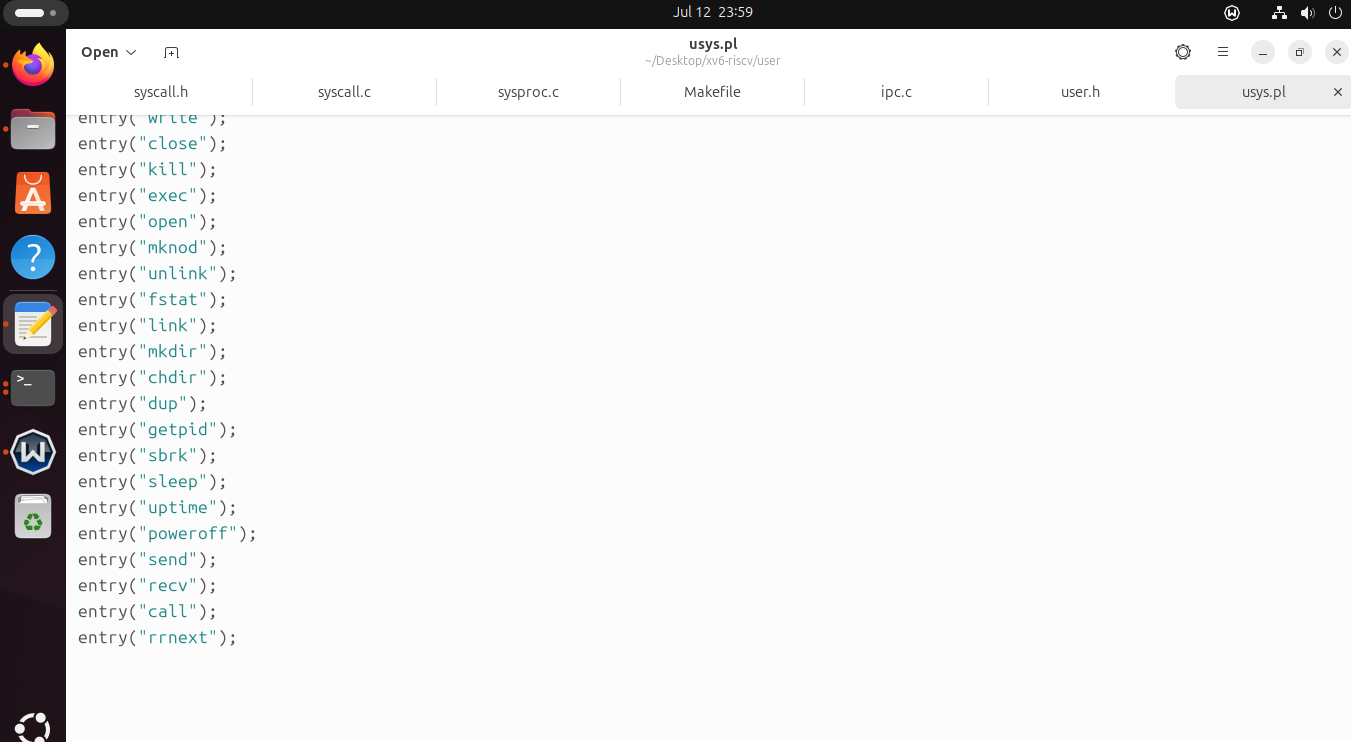
**توضیحات خط به خط کد:**

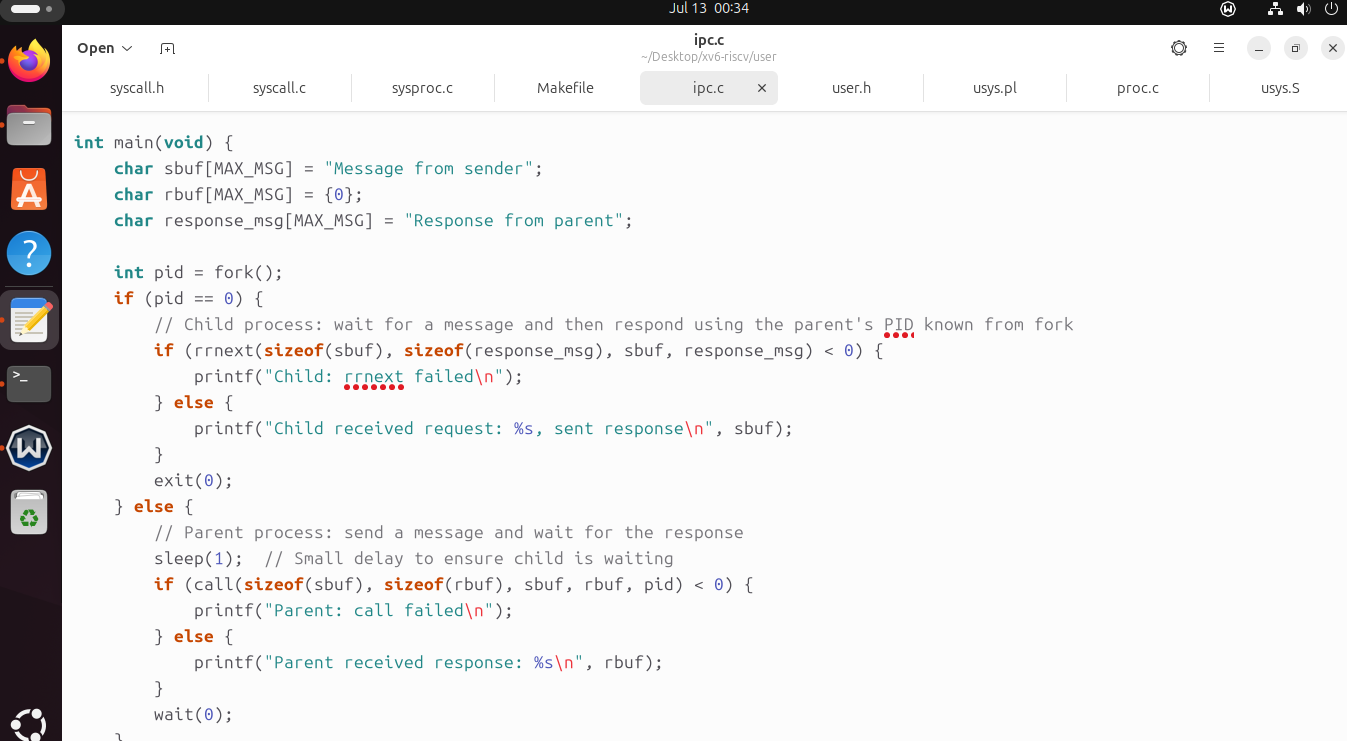
1. int sys\_rxnext(void) {:
   * تعریف تابع sys\_rxnext که هیچ آرگومان ورودی نمی‌پذیرد.
2. int sbuf\_len, rbuf\_len;:
   * تعریف متغیرهای محلی برای طول بافرهای ارسال (sbuf\_len) و دریافت (rbuf\_len).
3. char \*sbuf, \*rbuf;:
   * تعریف اشاره‌گرهای محلی برای بافرهای ارسال (sbuf) و دریافت (rbuf).
4. argint(0, &sbuf\_len);:
   * استخراج آرگومان اول (طول بافر ارسال) از سیستم‌کال و ذخیره آن در sbuf\_len.
5. argint(1, &rbuf\_len);:
   * استخراج آرگومان دوم (طول بافر دریافت) و ذخیره آن در rbuf\_len.
6. argaddr(2, (uint64\*)&sbuf);:
   * استخراج آرگومان سوم (آدرس بافر ارسال) و ذخیره آن در sbuf.
7. argaddr(3, (uint64\*)&rbuf);:
   * استخراج آرگومان چهارم (آدرس بافر دریافت) و ذخیره آن در rbuf.
8. if (sbuf\_len < 0 || rbuf\_len < 0 || sbuf == NULL || rbuf == NULL) {:
   * بررسی اعتبار آرگومان‌ها و اگر هر کدام نامعتبر بود، -1 بازمی‌گرداند.
9. if (receive\_message(sbuf) < 0 || send\_message(rbuf\_len, rbuf, 0) < 0) {:
   * فراخوانی توابع receive\_message برای دریافت پیام و سپس send\_message برای ارسال پیام و اگر هر کدام با خطا مواجه شدند، -1 بازمی‌گرداند.
10. return 0;:
    * بازگرداندن 0 در صورت موفقیت‌آمیز بودن تمامی مراحل.

**تفاوت‌های اصلی با کد قبلی:**

1. در این کد، تابع receive\_message قبل از تابع send\_message فراخوانی می‌شود.
2. تابع send\_message در این کد آرگومان pid ندارد و به جای آن 0 به عنوان آرگومان سوم استفاده شده است.







در ادامه برای تست آن لازم بود تا وارد فضای کاربر شویم و موارد دلخواه خودمان را تعریف کنیم که در ipc.c از قبل وجود داشت تنها لازم بود تا آن و مواردی در usys و user.h را تغییر دهیم تا بتوانیم از سیستم کالهای جدید در فضای کاربری استفاده کنیم :

**توضیحات خط به خط کد:**

1. int main(void) {:
   * تعریف تابع main که نقطه شروع اجرای برنامه است.
2. char sbuf[MAX\_MSG] = "Message from sender";:
   * تعریف یک بافر (sbuf) برای پیام ارسال شده از فرآیند والد به فرآیند فرزند و مقداردهی اولیه آن.
3. char rbuf[MAX\_MSG] = {0};:
   * تعریف یک بافر (rbuf) برای دریافت پیام از فرآیند فرزند و مقداردهی اولیه آن به صفر.
4. char response\_msg[MAX\_MSG] = "Response from parent";:
   * تعریف یک بافر (response\_msg) برای پیام پاسخ که از فرآیند والد به فرآیند فرزند ارسال می‌شود و مقداردهی اولیه آن.
5. int pid = fork();:
   * ایجاد یک فرآیند فرزند با استفاده از fork و ذخیره شناسه فرآیند فرزند در pid.
6. if (pid == 0) {:
   * اگر pid برابر صفر باشد، کد مربوط به فرآیند فرزند اجرا می‌شود.
7. if (rrxnext(sizeof(sbuf), sizeof(response\_msg), sbuf, response\_msg) < 0) {:
   * فرآیند فرزند منتظر دریافت پیام می‌ماند و سپس پیام پاسخ را ارسال می‌کند. اگر خطایی رخ دهد، پیغام خطا چاپ می‌شود.
8. printf("Child received request: %s, sent response\n", sbuf);:
   * در غیر این صورت، فرآیند فرزند پیغام دریافتی را چاپ می‌کند.
9. exit(0);:
   * پایان اجرای فرآیند فرزند.
10. } else {:
    * کد مربوط به فرآیند والد اجرا می‌شود.
11. sleep(1);:
    * یک تاخیر کوتاه برای اطمینان از این که فرآیند فرزند منتظر است.
12. if (call(sizeof(sbuf), sizeof(rbuf), sbuf, rbuf, pid) < 0) {:
    * فرآیند والد پیامی را ارسال کرده و منتظر دریافت پاسخ می‌ماند. اگر خطایی رخ دهد، پیغام خطا چاپ می‌شود.
13. printf("Parent received response: %s\n", rbuf);:
    * در غیر این صورت، فرآیند والد پیغام دریافتی را چاپ می‌کند.
14. wait(0);:
    * فرآیند والد منتظر اتمام فرآیند فرزند می‌ماند.
15. return 0;:
    * بازگرداندن مقدار 0 به منظور نشان دادن پایان موفقیت‌آمیز برنامه.

**تفاوت‌ها و ارتباط با کدهای قبلی:**

این برنامه یک نمونه کاربردی از سیستم‌کال‌های sys\_rxnext و sys\_call در سیستم‌عامل xv6 است که نحوه ارتباط بین فرآیندهای والد و فرزند را نشان می‌دهد. فرآیند فرزند منتظر دریافت پیامی از والد می‌ماند و سپس پاسخی ارسال می‌کند. فرآیند والد پیامی ارسال کرده و منتظر دریافت پاسخ می‌ماند.