1.

ایده پایه در پشت CMA این است که به برنامه MPI اجازه دهد که از طریق ارتباط چند گره ایی کپی یگانه از پیام را ایجاد کند اما در روش shared memory ما کپی دوبل را از پیام داشتیم تا در pipe قرار گیرد.

این امر با این مورد امکان پذیر می شود که به پردازه مقصد اجازه می دهد که یک آدرس و فضا از پردازه مبدا بگیرد تا یک کپی از حافظه به طور مستقیم از پردازه مبدا بگیرد و آنرا از ظریق یک سیستم کال مستقیما در در آدرس خودش قرار دهد.

موضوع بعدی این است که shared memory و pipes در کرنل موجودند و فعالیت می کنند و کرنل آنها را مدیریت می کند ، اما همانطور که بالاتر هم گفته شد در CMA مستقیما اطلاعات با پردازه منتقل می شود و این کار بدون کمک کرنل انجام می شود.

3.

قابلیت این کار وجود دارد .

باید mutex در حافظه ایی باشد که با پردازه ها به اشتراک گذاشته شده است. در این حافطه باید طوری مقداردهی اولیه انجام شود که بقیه پردازه ها هم به قفل دسترسی داشته باشند. برای این منظور باید همه mutex ها باید از PThread\_Process\_Shared استفاده کنند. با این ویژگی امکان باز و بسته کردن قفل به همه thread ها داده می شود.

این ویژگی نوعی از mutex ها است که Recursive Mutex نامیده می شود که یک شمارنده مثل semaphore دارد اما فقط برای thread است. به این معنی است که که اگر یک thread یک mutex را قفل کند و مدل recursive باشد ، همان thread می تواند آن mutex را بار ها و بار ها بدون نتیجه در بن بست قفل کند.

در صورتی که نگه داشتن قفل برای امن کردن اطلاعات خوب نیست .

2.

According to the man page as long as you write less than \*\*PIPE\_BUF\*\* the atomicity of the read and write operations are guaranteed. But the overall safety depends on the number of bytes we're trying to read or write, the file descriptor working in a nonblocking fashion or not, and the count of the writers. Now we go over four possible scenarios.

if the pipe is full write fails like the second scenario, setting errno to \*\*EAGAIN\*\* otherwise, 1 to n bytes may be written so we have to keep track of the return value from write to check bytes written. This is not safe also and data may get corrupted by other processes performing the write operation.